ESCOLA UNIVERSITÀRIA POLITÈCNICA
DE VILANOVA I LA GELTRÚ

ENGINYER TÈCNIC DE TELECOMUNICACIÓ
PLA 1972

Equips Perifèrics i Terminals

Pràctica 2

"Simulació d' una calculadora senzilla mitjançant un programa en ensamblador".

Lluís Gutiérrez Gonzalez

LLuís Casals Ibañez

Curs 96 / 97

UPC

UNIVERSITAT POLITECNICA DE CATALUNYA

Departament de

Matemàtica Aplicada

i Telemàtica

Equips Perifèrics i Terminals

Pràctica 2

"Simulació d' una calculadora senzilla mitjançant un programa en ensamblador".

Lluís Gutiérrez Gonzalez

LLuís Casals Ibañez

Curs 96 / 97

PRÀCTICA 2 1

INTRODUCCIÓ. 3

Objectiu 3

Descripció del programa. 3
Objectiu 3

Algorisme. 4

Disseny i programació. 6
Descripció de les rutines: 6

REALITZACIÓ D' UN PROGRAMA EN ENSAMBLADOR. 1

Fases d' un programa en ensamblador 1

Edició del mòdul font 1

Ensamblat del mòdul font 2

Enllaçat del mòdul objecte 2

Depuració del programa executable 3

ESTRUCTURA D' UN PROGRAMA EN ENSAMBLADOR. 1

Capçalera del programa. 1

Segment de dades. 1

Segment de STACK (PILA). 2

Segment de Codi (instruccions). 2

Programa de rutines 3

CONVERSIO DE BCD - BINARI. 1

Conversio de binari a ASCII 1

INTERFÍCIE PER A TECLAT HEXADECIMAL. 1

Descripció interfície 1

Material necessari 1

Hardware 1

Software 2
Taula 1. Correspondència entre línes d' estat i registre d' estat del port d' impressora 3

5. Realització 4

REALITZACIÓ I CALENDARI DE LA PRÀCTICA. 1

Realització de la pràctica 1

Consultes 2

Presentació de la pràctica 2

Avaluació 2

BIBLIOGRAFIA 1

ANNEX 1. 1

Llistat programa principal, rutines i utilitats. 1

Programa principal. 2

Subrutines principals, rutines. asm. 6

Rutina retard 20 mseg. 8

Entrada i sortida DOS. 9

ANNEX 2. 1

Programació en ensamblador per a 8086 1

Programació en ensamblador per a 8086. 2
Procés de creació d' un programa executable 2
Codificació de les instruccions 2
Contingut del mòdul font 4
Instruccions de llenguatge ensamblador 5
Directives de definició de símbols 6
Directives de definició de dades 6
Directives d' especificació de procediment 9
Directives d' especificació de segment 10
Directives de control de l' ensamblador 12
Altres directives 13
Operadors 16
Models de programes .EXE i .COM 16
Fitxer executable de tipus .EXE 16
Fitxer executable de tipus .COM 18
Aplicacions a la programació en ensamblador 19
Comunicació a través de la memòria mitjançant variables compartides. 19
Comunicació passant l' adreça dels paràmetres. 21

ANNEX 3. 1

Resum instruccions ensamblador. 1

ANNEX 4. 1

Taula ASCII. 1

Introducció.

Objectiu

A través del disseny d' un programa senzill es vol que l' estudiant tingui contacte amb un entorn de programació per a microprocessador de 16 bits, en aquests cas per a 8086.

La pràctica està formada per les fases normals del desenvolupament d' un programa:

- Edició del mòdul font.
- Ensamblat del mòdul font.
- Enllaçat del mòdul objecte.
- Depuració del programa executable.
- Realització del teclat hardware.

Aquest programa conjuntament amb el teclat ha d' ésser capaç d' actuar com una senzilla calculadora de butxaca, de manera que introduint dos operands i la operació a realitzar, el programa ha de calcular el resultat i visualitzar-lo per pantalla.

A més, caldrà construir un teclat adient al cas, segons les instruccions que detallarem a continuació.

Descripció del programa.

Objectiu

La pràctica consisteix en dissenyar un programa capaç d' acceptar dades provinents d' un teclat de 16 tecles (veure fig. 1); i fer l' operació escollida. El resultat del programa, així com les dades d' entrada s' hauran de visualitzar per pantalla.

7	8	9	+
4	5	6	-
1	2	3	*
CI	0	=	/

Fig 1. Teclat calculadora.

L' usuari introduirà les dades de la manera següent:

Num1 oper *Num2* = resultat.

Les dades d' entrada s' aniràn visualitzan per pantalla, en el moment que l' usuari hagi premut la tecla corresponent. L' usuari premerà les tecles seguents:

digits: 0 .. 9 .

operacions: +, -, *, / i =.

El programa un cop hagi detectat el signe =, operarà amb les dades introduïdes previament i un cop calculat el resultat farà la conversió a ASCII, i el treurà per pantalla.

Num1 i *Num2* corresponent als operands de la operació a realitzar i poden tenir un màxim de dues xifres, amb zeros a l' esquerra, etc. Els operands son numeros sencers, sense xifres decimals. Les operacions a realitzar també donaràn un resultat sencer, no tenint en compte els decimals en cas de divisions.

Per al control del programa cal tenir en compte:

Si el nombre de tecles pitjades (abans de prémer el signe =) es > 5 caldrà donar un error en pantalla, i iniciar de nou el procediment.

Si s' ha premut la tecla ***Clear*** iniciarem de nou el procediment indicant que l' usuari ha premut voluntariament aquesta tecla.

Si la seqüència no correspon a les instruccions donades (més d' un signe d' operació), o altres tipus de malfuncionament, també donarem un missage d' error i inciarem de nou el pocediment.

Algorisme.

La seqüència d' operacions a realizar pel programa és:

Neteja pantalla i missatge benvinguda.

Missatge inicial.

Entrada dades.

Crida subrutina entrada tecla: ***Tecla***. Detecta la tecla pitjada (fila i columna).

Crida subrutina conversió tecla: ***Conversio***. Converteix la tecla pitjada al seu valor binari de 0 a 15.

Les dades entrades s' aniran emmagatzemant a memòria, en la variable ***entrada*** tipus byte i de llargada 6.

A més, convertirem la tecla (dígit o signe) a ASCII i la visualitzarem per pantalla, mitjançant la rutina ***Visual***.

Control d' entrada.

Si **# tecles pitjades > 6** cridar subrutina Error (error = 1), i anirem al final .

Si la tecla pitjada es ***CLEAR*** donar missatge a l' usuari (error = 3), i anirem al final.

Procés entrada.

Aquest procés el realitzarà la subrutina ***Procés***

Caldrà processar les dades entrades emmagatzemades a ***entrada*** :

L' operand1 es convertirà de BCD a binari, (cal tenir en compte només té dues xifres), i es guardarà a la variable **oper1** tipus byte.

L' operació a realitzar caldrà guardar-la a memòria en la variable ***operacio*** de tipus byte segons el codi següent:

`+ : 1 , - : 2, * : 3` i finalment `/ :4.`

Amb l' operand 2 procedirem com amb l' operand 1 i el guardarem finalment a la variable **oper2** de tipus byte.

Finalment si es detectés que no s' han introduït les dades de forma correcta caldria cridar a la subrutina error (error = 2), i anar al final.

Calcul de resultats.

La subrutina ***Calcul*** s' encarregarà d' operar amb oper1 i oper2, i un cop obtingut el resultat de l' operació, aquest s' emmagatzemarà a memòria en la variable ***resultat*** de tipus word.

Sortida de resultats.

La subrutina ***Sortida*** ens ha de permetre visualitzar el resultat. Aquest s' ha de convertir de binari a BCD i posteriorment cada digit a ASCII. Un cop convertits s' hauràn de visualitzar per pantalla.

Final.

La part final del programa farà sortir el missatge de comiat. L' usuari pot triar entre continuar (introduint un 0), o finalitzar (introduint un 1). Qualsevol altra tecla serà considerada errònia.

Disseny i programació.

El programa principal s' inclou en aquesta memòria, l' estudiant només caldrà que dissenyi i provi les rutines esmentades a continuació, segons les instruccions que detallarem en aquest apartat. El programa principal s' anomenarà ***prog2.asm***. Aquest programa es pot veure en l' annex 1.

Previament es molt interessant que l' estudiant s' asseguri de compendre perfectament el que s' ha de realitzar en aquesta pràctica. Les rutines que ha de fer cal programar-les en un mòdul apart, que anomenarem ***rutines.asm.***

Descripció de les rutines:

Tecla. Explorar i detectar la tecla pitjada per l ' usuari.

Input: no n' hi ha.

Output: AH: número de fila pitjada de 0 a 3 binari.

AL: número de columna pitjada de 0 a 3 binari.

La forma d' explorar el teclat es pot veure en el capitol xx.

Conversio: Converteix les coordenades de la tecla pitjada a un valor binari.

Input: Coordenades de la tecla pitjada: fila (AH) i columna (AL).

Output: A CL tindrem el resultat de la conversió (4 bits) segons la taula seguent :

digit **0 .. 9** : binari de 0 a 9.

Signe: **+** : 10; **-** : 11; ***** : 12; **/** : 13; **=** : 14 i **Clear** : 15 .

Visual: Rutina per visualitzar un dígit o un signe per pantalla, s' ha de convertir el codi binari (0..15) a ASCII.

Input: CL: Número (0..9), signes del 10 a 15 binari (4 dígits).

Output: dígit a pantalla.

Procès: Processa ***entrada*** per obtenir ***oper1, oper2 i operacio.***

Input: variable ***entrada*** de memòria conté les dades d' entrada.
Output: ***oper1***, ***oper2*** i ***operació***. El retorn serà AL = 0, si les dades son correctes; en cas contrari AL = 0FF h.

Càlcul: Realitza el càlcul de les dades anteriors, el resultat s' emmagatzema a ***resultat***.

Input: ***oper1, oper2 i operacio***.

Output: ***resultat***.

Sortida: Visualització del resultat obtingut, cal convertir el resultat binari de ***resultat***, a BCD desempaquetat i posteriorment cada digit convertir-lo a ASCII per visualitzar-lo per pantalla. Consulteu el capitol xx.

Input: ***resultat.***

Output: Número a pantalla.

Error: Com hem destacat tenim fins a 3 possibles errors, caldrà donar un missatge a l' usuari indicant l' error comès, inclourem aquí el cas de pitjar la tecla ***Clear***.

Input: AL : nombre del error (1 .. 3).

Output: Missatge d' error a pantalla.

Realització d' un programa en ensamblador.

Fases d' un programa en ensamblador

A través del disseny d' un programa senzill es vol que l' estudiant tingui contacte amb un entorn de programació per a microprocessador de 16 bits, en aquests cas per a 8086.

La pràctica està formada per les fases normals del desenvolupament d' un programa:

- Edició del mòdul font.
- Ensamblat del mòdul font.
- Enllaçat del mòdul objecte.
- Depuració del programa executable.

Edició del mòdul font

Quan ja es disposi d' un disseny del mòdul font prou fiable es passarà a la fase d' edició, on l' objectiu és crear un fitxer de text que contingui el mòdul font.

Es fa servir un editor amb sortida ASCII (sense codis de control específics del processador de text) com pot ser l' editor de l' entorn de programació de Turbo Pascal o Turbo C, o l' editor del sistema operatiu MS-DOS *edit.exe*. S' ha de tenir en compte que a l' hora de donar el nom del fitxer font la seva extensió sigui *.ASM*.

L' editor del sistema operatiu *edit.exe* funciona en base un conjunt de menús que són accessibles prement i deixant anar la tecla ALT. Un cop situats a la barra de menú (un dels menús apareixerà en vídeo invers) ens podem desplaçar cap a altres menús amb les tecles de cursor i desplagar-los prement la tecla RETURN. Dins d' un menú desplegat podem escollir una opció despleçant-nos amb les tecles de cursor i prement RETURN quan l' opció que interessa estigui en vídeo invers.

La pràctica que estem realitzant constarà de dos programas independents tal com ja s' ha explicat, aquest son:

- programa principal: *prog2.asm.*
- rutines: *rutines.asm.*

A l' hora de treballar des del punt de vista d' edició, cal pensar en dos programes independents, per bé que a l' hora d' enllaçar als mòduls, aquest formin part d' un programa final comu ***prog2.exe***.

Ensamblat del mòdul font

Un vegada realitzat el mòdul font cal que sigui traduït a instruccions vàlides per al 8086. Per això es farà servir un ensamblador per a 8086, com pot ser el Turbo Assembler.

Per ensamblar un programa s' ha d' executar l' ordre:

```
TASM  /zi  nom_fitxer_font,,
```

`nom_fitxer_font` és el nom del fitxer que conté el programa font. No cal especificar la extensió si la que té el fitxer és *.ASM*.

`/zi` és un paràmetre que permet generar informació addicional per facilitar la depuració.

Si s' executa l' ordre TASM sense cap paràmetre, es mostra la sintaxi de l' ordre d' ensamblat i una llista dels possibles paràmetres.

Com a resultat de l' ensamblat s' obté un fitxer de codi objecte amb el nom `nom_fitxer_font` i extensió *.OBJ* que conté el programa font traduït a codi màquina. Aquest fitxer de codi objecte encara no serà executable doncs s' han de concretar les referències a posicions de memòria que s' han resolt de forma relativa, o sigui, s' han donat uns valors relatius al començament del segments definits en el mòdul que conté el programa font. S' estableix aquesta forma de funcionament per poder crear codi objecte que es pugui enllaçar amb altres fitxers de codi objecte per formar un únic programa executable.

En el cas que el mòdul font contingués errors de sintaxi o qualsevol altre relacionat amb el llenguatge ensamblador, seran detectats durant aquesta fase i haurem de tornar a editar el programa per corregir els error que l' ensamblador ha trobar. Recordeu que els errors que es produeixin en la fase d' enssamblar el programa, es veuen perfectament explicats en el fitxer * *.LST*, i es per tant, en aquest que hem de mirar tots els problemes que s' hagin produit. Cal però recordar que si s' ha de modificra el programa, això es fa en el font, és a dir, en el fitxer **.ASM*, que es ela que s' ha de compilar mitjançant el Turbo-Assembler.

En l' annex 3 hi trobareu un resum de les instruccions i directives del llenguatge ensamblador per a 8086.

Enllaçat del mòdul objecte

Un cop s' ha obtingut un fitxer objecte correcte en passa a la fase d' enllaçat. Aquesta fase permet combinat diversos mòduls objecte per obtenir un únic programa executable. En aquesta fase es crea un fitxer que contindrà el codi màquina del programa que estem desenvolupant amb les referències a memòria concretades segons les posicions que ocuparà cada mòdul objecte en el programa final.

Per enllaçar el nostre programa s' ha d' executar l' ordre:

```
TLINK  /v  nom_fitxer_objecte
```

`nom_fitxer_objecte` serà el mateix que `nom_fitxer_font` si s' han seguit les indicacions anteriors.

`/v` és un paràmetre que fa que l' enllaçador generi informació addicional per facilitar la depuració.

Si s' executa l' ordre `TLINK` sense cap paràmetre ni nom de fitxer es mostra una línia amb la sintaxi de l' ordre i una llista de paràmetres possibles d' enllaçat.

Al final d' aquest procés s' obté un fitxer executable que té per nom nom_fitxer_objecte i extensió .EXE. Aquest fitxer conté un programa que es pot executar cridant-lo com si fos un ordre:

```
nom_fitxer_executable
```

En aquesta pràctica cal recordar que estem treballant amb dos mòduls; *prog2.obj* i *rutines.obj*, per tant a l' hora d' enssamblar els dos mòduls conjuntament hem de fer:

```
TLINK  /v  prog2+rutines
```

Un cop fet això tindrem el nostre programa executables per provar: **prog2 . exe**.

Malgrat això, es aconsellable fer l' execució des d' un depurador de programes tal com el Turbo Debugger, per poder analitzar els errors que poguin surgir.

Depuració del programa executable

Podem cridar al programa executable de la següent manera:

```
TD  nom_fitxer_executable
```

Si el fitxer especificat és un fitxer que s' ha generat amb informació addicional per a la depuració, en la pantalla inicial del programa de depuració es mostrarà una finestra anomenada Module on apareix el mòdul font en ensamblador.

Des d' aquest moment podem aplicar totes les funcions que ens permet el depurador per verificar el nostre programa des de la finestra Module. També podrem visualitzar zones de memòria d' interès, registres, modificar zones de memòria o registres, posar punts de Break, visualitzar variables, etc. Les funcions i característiques del depurador estan resumides en l' annex B.

Per comprovar el funcionament del programa, abans de fer la primera execució, farem que es visualitzi la zona de memòria on hi ha les variables del segment de dades. Això es pot fer de dues formes diferents:

- amb l' opció Dump del menú View.
- amb l' opció Watch del menú Data.

Per escollir un dels dos menús ens hem de situar en la barra de menú superior prement F10, i desplaçant-nos fins al menú desitjat amb les tecles de cursor. Una vegada situats sobre el nom del menú es desplega el menú prement RETURN. També podriem desplegar-lo directament si una vegada situats en la barra de menú es prem la lletra que té diferent color en el nom del menú desitjat.

Per escollir una opció determinada del menú es pot procedir de forma semblant a la descrita anteriorment pel cas del menú.

Si s' escull l' opció DUMP, es visualitzarà una finestra superposada a la finestra activa fins aquell moment, on es mostra el contingut de la zona de memòria que correspont a la base del segment assenyalat pel registre de segment DS. Abans de començar l' execució del programa DS no està apuntant al segment de dades que s' havia definit en el programa font sinò a una zona de paràmetres del sistema operatiu que crea cada cop que es carrega un programa a memòria per executar-lo. Aquesta zona, anomenada PSP (Prefixe de Segment de Programa), ocupa els 256 bytes primers de la zona de memòria del sistema on es carreguen els segments de dades, de codi i de pila del programa a executar.

Per tant, si ens desplacem amb el cursor a través de la zona de memòria que es visualitza fins a la posició DS:100 veurem la zona de dades del nostre programa (sempre que el segment de dades sigui el primer segment declarat en el nostre fitxer font).

En el cas que les instruccions que inicialitzen els registres de segment ja haguessin estat executades abans d' escollir l' opció DUMP (o l' opció CPU), la zona de memòria que es mostra correspon directament al nostre segment de dades (indicada amb l' adreça lògica DS:0000).

Si s' escull l' opció Watch, es donarà directament el nom de la variable que es vol visualitzar.

Pot passar que quan es vulgui tornar a la finestra de Module la finestra de Dump quedi a sota d' aquella i no es vegi la zona de memòria escollida. Per poder veure les dues finestres a l' hora cal fer un moviment, i potser també un ajustament del seu tamany, amb les opcions de Move/Resize... del menú Window. Per altra part, amb la opció Window pick d' aquest mateix menú podrem canviar la finestra activa.

Una cop tenim visibles les zones de dades que ens interessa per tal de comprovar si el resultat de l' execució és correcta, podem passar a fer aquesta execució. La podrem fer de diverses formes:

- Execució completa: Opció Run del menú Run (o directament F9)
- Execució pas a pas: Opció Trace Into del menú Run (o directament F7)
- Execució en animació: Opció Animate del menú Run (o directament Alt-F4)
- Execució fins el cursor: Opció Go To Cursor del menú Run (o directament F4)

i altres funcions que vosaltres mateixos podeu experimentar.

Serà interessant, potser, posar punts de Break mitjançant les opcions del menú Breakpoint, per fer aturar l' execució en punts determinats del programa i així poder inspeccionar el contingut dels registres, zones de memòria, variables, etc., per facilitar la localització de possibles errades en el programa dissenyat.

Amb l' opció CPU del menú VIEW podreu tenir una altra visió del programa a depurar, que també pot facilitar la feina de comprovació del programa. En aquest cas, es mostra una finestra amb diverses parts on apareixen el programa en codi màquina i el desensamblat corresponent, els registres del microprocessador, els registres de flags, la pila i la zona de memòria apuntada pel valor del registre DS. Cada part d' aquesta finestra té un menú local accessible amb Alt-F10. Per desplaçar-se per les parts de la finestra es farà servir la tecla de tabulació (Tab). Finalment, s' aconsella que exploreu pel vostre compte la resta de possibilitats que ofereix l' entorn de programació utilitzat per a la pràctica.

Estructura d' un programa en ensamblador.

En aquest capitol donem les idees principals de la programacció en ensamblador. Cal tenir en conmpte que podeu trovar el llistat complet de ***prog2. asm***, i també part de ***rutines.asm*** a l' *Annex 1*.

Un programa en enssamblador té una estructura fonamental que cal repectar. Les diferents seccions d' un programa les veiem en les apartats seguents d' aquest capitol.

- Capçalera del programa.
- Segment de dades. Conté les variables de memòria que farà servir el programa.
- Segment de stack. Conté la zona stack del programa.
- Segment de Codi. En el que hi ha el programa (instruccions), i rutines formant un o varis Pocediments PROC.

Capçalera del programa.

En la capçalera del programa hi han una sèrie de pseudo-instruccions que permeten definir uns quants paràmetres, alguns d' ells necessaris a l' hora de fer funcionar definitivament el programa, i d' altres que ens fan aquest més intel.ligible:

TITLE	Permet anomenar el nostre programa
ORG	Establim un origen fix del programa.
EXTRN	Definim el nom dels mòduls externs.
PUBLIC	Permet definir etiquetes publiques per tots els mòduls.

Segment de dades.

En el nostre cas ja s' ha comentat les variables mínimes necessaris per realizar el programa, que són:

```
DADES   SEGMENT
                ; definició de les variables del programa
    OPER1    DB     1        ; OPERAND 1 (BYTE)
    OPER2    DB     1        ; OPERAND 2 (BYTE)
    RESULTAT        DW1      ; RESULTAT (WORD)
    ENTRADA  DB     6 DUP(0) ; ENTRADA DADES (ARRAY OF BYTES)
```

```
NBCD      DB      5 DUP(0) ; resultat de la conversió a BCD
                   desempaquetat
ASBCD     DB      5 DUP(' ') ; resultat de la codificació
                   BCD desempaquetat a ASCII
RESPOSTA          DB?       ; RESPOSTA FINAL USUARI 0 - NO, 1
                   SI
CTRL_SCR          DB'    ' ; NETEJA PANTALLA
MISS_IN  DB       ' EPT - PRACTIQUES-96  -  PRACTICA DE LA
                   CALCULADORA $'
MISS_END DB       ' VOLS ENTRAR MES VALORS ?  (0 - NO, 1 -
                   SI ): $'
MISS_FIN DB       ' EPT - PRACTIQUES-96  -  A REVEURE FINS
                   UNA ALTRA OCASIO $'
MISS_LINE         DB0AH,0DH,'$'      ; SALT DE LINIA (cr , .
                   lf)

DADES   ENDS
```

A part d' aquestes definicions, en el moment de fer el programa es poden afegir les variables que hom cregui necessàries.

Segment de STACK (PILA).

Com ja coneixeu l ' stack es una zona de memòria a on automàticament es guarden dades, per exemple en cas de CALLs a subrutines, etc. A més el programador també pot guardar-hi dades de forma temporal (PUSH i POP). Cal tenir sempre en compte que aquesta zona funciona en modus PILA, i que guardem i recuperem de forma homogenea.

La definició de l' stack en el nostre programa serà com esmentem a continuació:

```
PILA   SEGMENT   STACK
        DW  128   DUP 'S'        ;definició de la pila
del programa
PILA   ENDS
```

Això fa que el nostre programa contingui una zona de stack de 128 WORDS (256 bytes), i que podem reconèixer fàcilment per la seva inicialització al caràcter 'S'.

Segment de Codi (instruccions).

El segment de codi conté les instruccions del programa, i sempre ha de respetar la estructura que veiem a continuació.:

```
EXTRN TECLA:NEAR,CONVERSIO:NEAR,PROCES:NEAR,CALCUL:NEAR
        .    .    .    .
CODI     SEGMENT
ASSUME  CS:CODI,DS:DADES,SS:PILA
```

```
PRINCI   PROC
ENTRADA: MOV  AX,DADES ; punt d' entrada al
         MOV  DS, AX   ; programa i
         . . . .       ;...
         . . . .       ; inicialització del
            ;registres de segment
         . . . .       ; instruccions del programa
principal
PRINCI   ENDP

PEPE     PROC
         . . . .       ; subrutina PEPE del programa
         RET
PEPE     ENDP

JOAN     PROC
         . . . .       ; subrutina JOAN del programa
         RET
JOAN     ENDP
                       ; altres subrutines
CODI  ENDS
END  PRINCI            ; final del mòdul font i

                 ; indicació punt entrada al programa
```

(les directives d' ensamblador apareixen en negreta)

Programa de rutines

Per a dissenyar el programa de rutines hem de seguir l' esquema seguent:

```
PUBLIC        RUTINES ; declara símbols públics
EXTRN         OPER1:BYTE, OPER2:BYTE, RESULTAT:WORD, . . .
                                .
DADES         SEGMENT   PARA   PUBLIC
              ...
DADES         ENDS

PILA          SEGMENT   STACK
              ...
PILA          ENDS

CODI_RUTINES SEGMENT
              ASSUME   CS:CODI, DS:DADES, SS:PILA
TECLA         PROC
              ...            ; instruccions  rutina
              RET
TECLA         ENDP
CONVERSIO     PROC
              ...
              ...            ; instruccions  rutina
              ...
              RET
```

```
CONVERSIO     ENDP
ENTRADA       PROC
              ...
              ...              ; instruccions  rutina
              ...
              RET
ENTRADA       ENDP
PROCES        PROC
              ...
              ...              ; instruccions  rutina
              ...
              RET
PROCES        ENDP
CALCUL        PROC
              ...
              ...              ; instruccions  rutina
              ...
              ...
              RET
CALCUL        ENDP
              ...
              ...              ; altres rutines
              ...
CODI_RUTINES      ENDS
              END
```

Conversio de BCD - binari.

Quan l' usuari introdueix xifres per ser operades, introdueix 1 o varis números a través del teclat, per exemple si vol introduir el 45, introduirà un 4 i el 5, que restaràn emmagtzemat a la variable ***entrada***, en dos bytes contigus però independents.

Per poder operar amb aquestes xifres com un número cal convertir-les a binari pur, això es un procès molt senzill, i consisteix en dir que:

```
Num =  xifra1 * 10 + xifra 2;
```

El resultat d' aquesta operació ja és el número binari en forma operable, i que en el nostre cas guardarem a memòria, variable ***operx***.

Conversio de binari a ASCII

El nostre programa calcula un resultat que el guardarem a memòria en la variable ***resultat*** tipus WORD. Això vol dir que tenim un número binari que pot anar desde le valor 0 (0 .. 0), fins al valor 65535 (0FFFFH).

Per presentar aquest resultat a pantalla caldrà convertirlo a BCD desempaquetat, (5 xifres decimals); deixant aquest resultat intermig a ***nbcd***, i posteriorment convertir cadascun dels digits binaris BCD a ASCII, deixant aquest resultat a ***ascbcd***.

```
RESULTAT        DW1       ; RESULTAT (WORD)
NBCD     DB     5 DUP(0)  ; resultat de la conversió a BCD
                           desempaquetat

ASBCD    DB     5 DUP(' ') ; resultat de la codificació
                           BCD desempaquetat a ASCII
```

Imaginem que a ***resultat*** tenim un valor binari pur:

10001011 01001100

Aquest valor és el 35660 si el convertim a decimal. Per fer la conversió del valor binari al BCD, caldrà dividir el número binari original per 10 (fins a 5 cops), i anar recollint els residus que seràn les xifres BCD. Per tant, tindrem com a resultat 5 bytes BCD, que vol dir que cada dígit és una xifra decimal: decenes de mil, milers, centenes, ...; en el nostre cas:

nbcd :

00000011 00000101 00000110 00000110 00000000 (=> 35660 decimal)

Aquest resultat ja té 5 xifres, però no és visualitzable perquè el contingut és binari, per poder visualitzar-les definitivament caldrà convertir cadascuna d' elles al codi ASCII (sumant 30H), també podem utilitzar la rutina ***Visual***. El resultat ASCII l' emmagatzemem a ***ascbcd***.

Interfície per a teclat hexadecimal.

Descripció interfície

En aquesta pràctica es vol dissenyar una interfície (hardware i software) de teclat hexadecimal per a PC. El disseny constarà de dues parts. Una part hardware que adaptarà el dispositiu físic al port de la impressora de l' ordinador, i una part software que permetrà utilitzar el dispositiu físic d' una forma còmoda estalviant la necessitat de conèixer els detalls elèctrics i mecànics del perifèric.

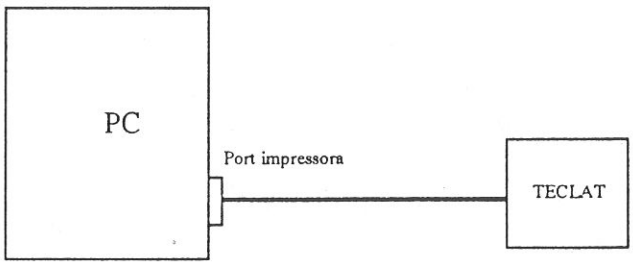


Figura 2. Interfície física per a teclat hexadecimal

Material necessari

Per a la realització de la pràctica serà necessari el següent material:

- 16 tecles (pulsadors) per a teclat (que siguin barates: 50 - 100 ptes).
- 10 metres de cable rígit de 0,3 mm de diàmetre.
- Un connector tipus D (Canon) mascle de 25 contactes.
- 4 díodes (1N4004), 4 resistències de 10 Kohms.
- Alicates, soldador i estany.

Hardware

La interfície hardware haurà d' adaptar els aspectes físics (elèctrics i mecànics) del PC i del teclat. La comunicació amb el teclat es farà a través del port paral.lel de la impressora. En aquest port disposem de 8 línies de sortida de dades de les quals només utilitzarem les 4 de menor pes com a terminals de sortida i es faran servir 4 senyals d' estat del mateix port com a terminals d' entrada. L' esquema de connexió es mostra a la figura 2. Les quatre línies de sortida i d' entrada seran accessibles a través d' un connector tipus D de 25 pins (mascle).

És important tenir en compte que en la solució de disseny que es proposa es considera que no es donarà mai la possibilitat que hi hagi més d' una tecla activada. En especial cal que no s' activin de forma simultània tecles de diferents files, ja que podria provocar danys en el hardware del port de la impressora del PC. Per evitar aquest perill, s' inclouen els díodes en les línies de dades de forma que ja no es podrà donar un curt circuit entre dues sortides de dades.

Perquè tot funcioni correctament, també cal assegurar que les línies d' entrada d' estat tinguin, per defecte, un nivell lògic inactiu, la qual cosa s' aconsegueix imposant un 1 a aquestes línies a través d' una resistència. Aquest fet condiciona la forma d' explorar el teclat que consistirà en detectar zeros en els terminals d' estat. Per poder assegurar l'1 a les línies d' estat utilitzarem una línia de sortida de control (SELECT IN) que serà posada a nivell 1 en la inicialització del programa (tal com s' explica en l' apartat 4).

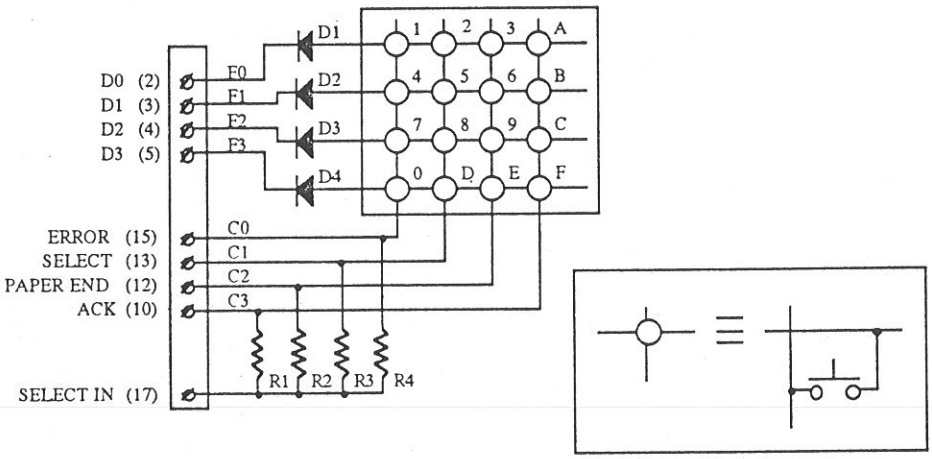


Figura 3. Esquema i connexionat del teclat

Software

Una vegada hem solucionat la interfície hardware haurem de desenvolupar un conjunt de rutines que permetin utilitzar el teclat d' una forma senzilla, ocultant, si és possible, els aspectes hardware del dispositiu que volem controlar.

Per això haurem de tenir en compte el següent:

- Es considera que no hi haurà vàries tecles accionades de forma simultània.
- El teclat es connecta al PC a través del port d' impressora. Aquest port té un registre de sortida de dades (registre de dades) del qual els bits D0 a D3 estaran connectats als terminals F0 a F3 del teclat.
- El port de la impressora també té un registre d' estat que guarda l' estat de determinades línies d' entrada. Nosaltres utilitzarem quatre d' aquestes línies que tenen correspondència amb els bits 3, 4, 5 i 6, tal com es mostra en la

taula 1. Al mateix temps, aquestes línies es connectaran al teclat a través dels terminals C0 a C3.

Taula 1. Correspondència entre línes d' estat i registre d' estat del port d' impressora

Senyal d' estat	Bit del registre d' estat
ERROR	Bit 3
SELECT	Bit 4
PAPER END	Bit 5
ACK	Bit 6

(El bit de menor pes és el bit 0)

a) El port d' impressora té un registre de control que determina l' estat dels senyals de control d' impressora. Només utilitzarem la línia SELECT IN que té correspondència amb el bit 3 del registre de control; aquest bit haurà de ser posat a nivell baix (0) per tal que la línia SELECT IN doni un nivell alt (1) i que s' utilitzarà per alimentar a les resistències de polarització, tal com s' ha dit anteriorment. També cal que el bit 4 es posi a nivell 0 per evitar que s' activin interrupcions d' impressora.

b) Figura 4. Algorisme d' una exploració del teclat. Per obtenir l' adreça del registre de dades del port d' impressora s' ha de llegir una paraula que hi ha a l' adreça física 408h (de memòria). L' adreça del registre d' estat i del registre de control s' obtenen incrementant en 1 i en 2, respectivament, el valor de la paraula llegida en aquella posició física.

c) Per detectar l' accionament d' una tecla es procedirà de la manera següent:

3) S' excitarà una línea de fila amb un 0 (mantenint la resta de línies de fila a nivell 1) i es comprovarà en quina columna hi ha un 0. Si es detecta una columna amb un 0 voldrà dir que la tecla situada en el creuament de la fila excitada i la columna detectada ha estat pitjada. Es repetirà aquest pas per a cada fila fins que es detecti una columna activada (a 0).

4) Quan es detecti una columna activa s' haurà d' esperar 20 ms per deixar passar el transitori de la connexió de la tecla i evitar possibles errors en la lectura del seu estat. Per produir aquest retard es pot utilitzar la rutina de l' annex 1 (inclosa a ***rutines.asm)***.

5) Després del retard anterior comprovarem si la columna encara és activa i si ho és es donarà per bona l' activació de la tecla situada a l' encreuament fila-columna activada. A l' algorisme de la figura 3 es representa aquesta forma de procedir.

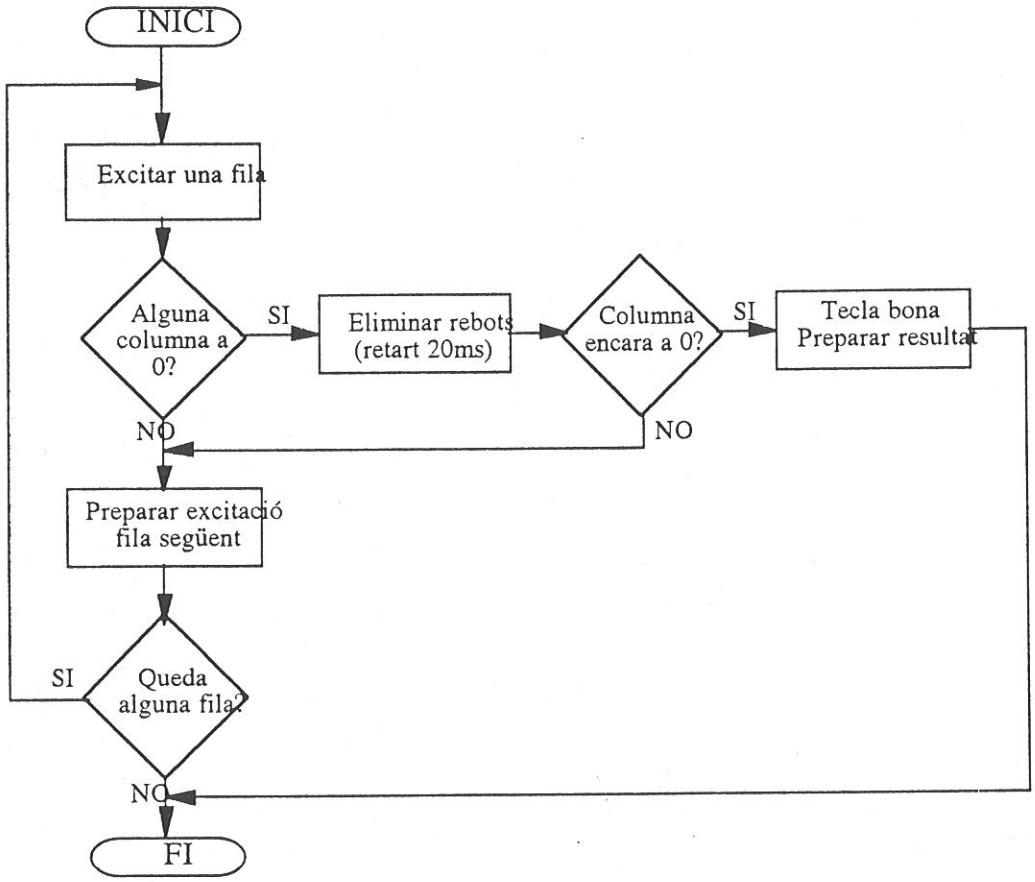


Fig. 4. Diagram de fluxe exploracio teclat.

5. Realització

Primer haurem de construir la interfície hardware tal com s' ha explicat anteriorment i **comprovar** amb molta cura que totes les connexions han estat fetes correctament.

Per construir el teclat es pot procedir de la forma següent:

a) Clavar les tecles en una cartulina o cartró, no massa gruixut però prou consistent, en el mateix ordre que apareixen en l' esquema.

b) Talleu 9 troços de cable de 100 cm i desprotegiu 10 cm d' un dels seus extrems.

c) Per la cara del teclat on hi ha els contactes de les tecles soldareu primer els cables que corresponen a les files (vegeu la figura 4).

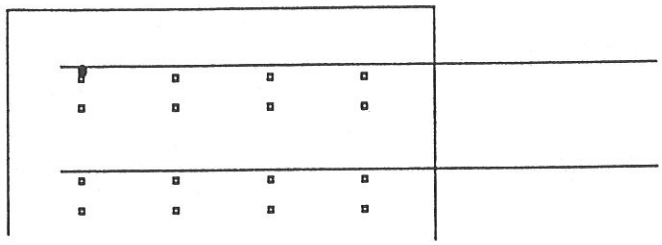


Figura 5. Connexió de les files del teclat

d) Després soldareu un cable per cada columna, donant al cable una forma de pont entre contacte i contacte de tecla per evitar la connexió elèctrica amb els cables de les files (vegeu figura 5).

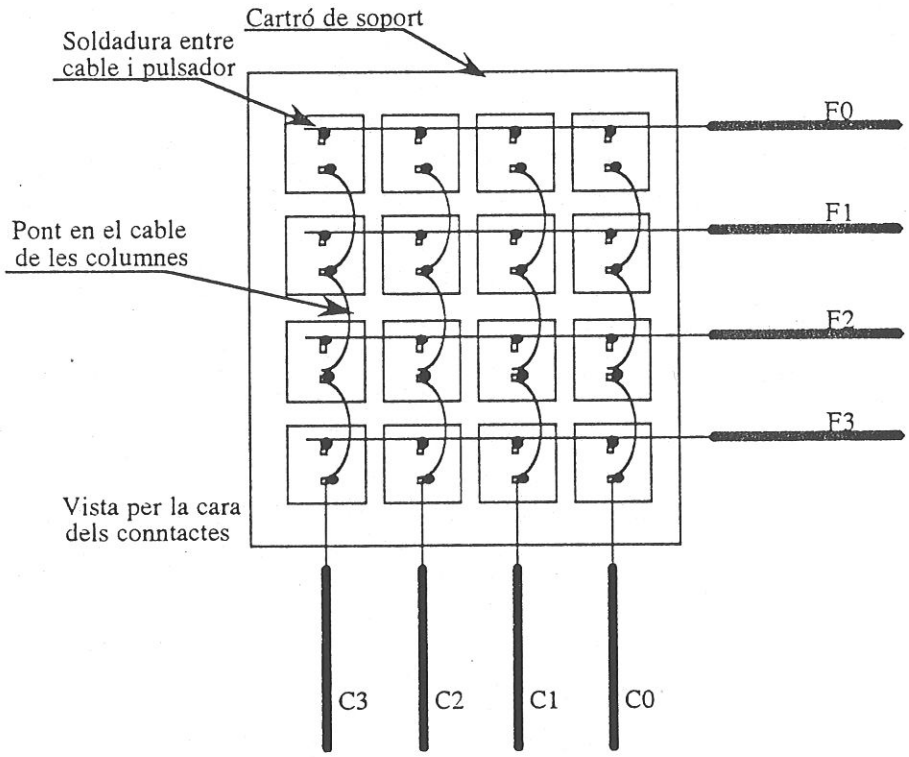


Figura 6. Connexió de les columnes del teclat

e) Soldar l' extrem lliure de cada cable en el pin que correspongui del connector tipus D.

Després, dissenyarem el software tenint en compte l' algorisme que s' ha explicat i els següents pasos:

a) Dissenyar La rutina (TECLA) que explori el teclat fins que detecti l' accionament d' una tecla. Aquesta rutina ha de donar com a resultat els següents valors en AX:

AH: ha de tenir tots els seus bits a 0 excepte el bit que correspont a la fila de la tecla detectada. La correspondència de bits i files és: bit0 - F0, bit 1 - F1, ..., bit 3 - F3.

AL: ha tenir tots els bits a 0 excepte el bit corresponent a la columna que s' ha detectat a 0 que s' haurà de posar a 1. La correspondència de bits i columnes és: bit 0 - C0, ..., bit 3 - C3.

Per detectar la polsació de les tecles correctament, aquesta rutina basarà el seu funcionament en la repetició dels dos passos següents:

Primer: ha de fer exploracions del teclat fins que obtingui una exploració sense cap tecla activada.

Segon: ha de fer exploracions del teclat fins que obtingui una tecla polsada vàlida.

b) Fer una rutina (CONVERSIO) que doni com a resultat, en AL, el valor binari de la tecla accionada, a partir dels valors de fila i columna que rebrà de la rutina través d' aX.

c) Ambdues rutines formaràn part del modul rutines. Els procediments anteriors formaran part del mòdul que anomenem *rutines.asm*. Tindrà definit un segment de dades, pila i codi que haurà de seguir l' esquema següent, utilitzant els mateixos noms de segment que es donen, tal com s' ha vist en el capitol xx.

Realització i calendari de la pràctica.

Per a la realització de la pràctica es proporcionen els següents programes:

- Editor: EDIT.EXE, editor del sistema operatiu, en el directori \DOS
- Ensamblador: TASM.EXE, en el directori \TASM.
- Enllaçador: TLINK.EXE, en el directori \TASM
- Depurador: TD.EXE, en el directori \TD
- Programes ***prog2.asm*** i ***rutines .asm***, en el directori \TASM

Aquests programes es proporcionaran en la sessió de la pràctica d' iniciació.

Com ja es va comentar en aquesta sessió cal que l' estudiant observi les normes següents:

En totes les sessions de pràctiques l' estudiant o estudiants d' un grup han de dur el diskette en el que hi ha els programes esmentats, més els que ell o ells hagin realitzat.

Cal que tots els programes realitzats pel estudiant o estudiants resideixin en el directori A:\ PRAC.

Realització de la pràctica

La pràctica es realitza en la sala de PC's i és obligatori que tothom assisteixi a totes les sessions, de dues hores cadascuna, de que consta la pràctica. La pràctica es realitzarà en grups de dues persones i un grup per ordinador.

En la pràctica d' iniciació s' expliquen tots els conceptes fonamentals i els procediment d' operació bàsics necessaris per a la realització de les pràctiques.

El treball a realitzar es farà en quatre sessions, cadascuna d' elles l' estudiant ha d' assolir uns resultats que indiquem en aquest punt. Cal que l' estudiant segueixi de forma acurada el calendari proposat en aquest apartat.

> Primera sessió: Portar un esborrany del programa seguint l' esquelet que es dóna en l' enunciat i dissenyar detalladament les rutines ***Visual*** i ***Sortida***.
>
> Segona sessió: Caldrà portar el teclat segons s' explica en el Capitol xx. A més es desenvoluparàn la rutina ***Tecla.***

Tercera sessió: En aquesta sessió finalitzarem les rutines principals de *Proces* i *Calcul*.

Quarta sessió: En aquesta sessió es farà l' evaluació final del treball de l' alumne. La forma d' avaluar està indicada en aquest capítol.

Consultes

Durant les hores de pràctiques un professor de l' assignatura estarà present en la sala de PC's per atendre els dubtes que sorgeixin en la realització de la pràctica. A més l' estudiant pot emprar les hores de consultoria dels professors de l' assignatura LLuís Casals i LLuís Gutierrez, que restaran disponibles en el seu despatx D114, (zona 4 desp. 2).

Presentació de la pràctica

Per cada parella s' haurà d' entregar un informe amb el llistat del programa.

A més aquest informe, haurà d' explicar el següent.

Grups *A, B, C, D, E*: L' informe ha de consistir en un article per a una revista de PC's tipus Byte, etc., en el que es descrigui com funcionaria una calculadora feta segons la pràctica realitzada. L' article redactat ha de ser de 3 pàgines com a màxim i ha d' explicar de forma didàctica, entenedora i engrescadora el treball realitzat.

Grups *F, G, H, I, J i K*: L' informe ha de consistir en una exposició del funcionament bàsic d' un PC, així com posar d' exemple el treball realitzat. Cal redactar l' article pensant en una revista d' Institut, caldrà doncs explicar molt didàctica i fàcilment com funcionaria una calculadora feta segons la pràctica realitzada. L' article redactat ha de ser de 3 pàgines com a màxim.

Es farà un control de l' assitència a les sessions de pràctiques, sent obligatori assistir a totes les sessions. En alguna de les sessions es pot fer una avaluació (control) de la pràctica i del que l' alumne ha anat progressant, aquest control pot ser oral o escrit.

La data màxima d' entrega de l' informe final es el dia 29 de Maig fins les 18 h.

Les notes de la pràctica sortiran el dia 12 de Juny. Si fos necessari per questió d' examens sortirian abans en la data que ja es comunicarà oportunament.

Avaluació

En l' avaluació de la pràctica es tindran en compte els següents paràmetres (en ordre de major a menor prioritat):

- Respostes de l ' alumne a les qüestions, i proves parcials o final.
- Assistència a les sessions de les pràctiques.
- Realització de la feina prèvia a cada sessió.
- Comprensió del procés de desenvolupament d' un programa en ensamblador.
- Coneixements del funcionament i possibilitats del depurador.
- Cumpliment de les especificacions del problema plantejat.
- Informe presentat.

Bibliografia

[CAS95] Casals, Lluís
EQUIPS PERIFÈRICS I TERMINALS.
Edicions UPC. 1995.

[LIU86] Liu, Yu-Cheng; Gibson, G.
ARQUITECTURA, PROGRAMACION Y DISEÑO DE SISTEMAS BASADOS EN MICROPROCESADORES (8086/80186/80286).
Anaya, 1986.

[ROD] Rodríguez-Roselló, Miguel Angel
8088-8086/8087 PROGRAMACION ENSAMBLADOR EN ENTORNO MSDOS.

[PRO88] Protopapas, D. A.
Microcomputer Hardware Design.
Prentice-Hall International. 1988.

[UFF85] Uffenbeck, J.
Microcomputers And Microprocessors. The 8080,8085 and Z-80 programming, interfacing, and troubleshooting
Prentice-Hall International. 1985.

[LiG86] Liu, Yu-Cheng; Gibson, G
Arquitectura, programación y diseño de sistemas basados en microprocesadores (8086/80186/80286).
Anaya. 1986.

[Bel95] Beltran de Heredia, Jon
Lenguaje ensamblador de los 80x86.
Anaya. 1995.

Annex 1.

Llistat programa principal, rutines i utilitats.

Programa principal.

Codi font del programa principal de la pràctica 2, ***prog2 . asm***.

```
TITLE     Practica 2 EPT: Calculadora senzilla

PUBLIC    OPER1, OPER2, RESULTAT, ENTRADAD, NBCD, ASBCD, MISS_LINE,
        OPERACIO
PUBLIC    MISS_ERR1, MISS_ERR2, MISS_ERR3,

EXTRN    TECLA:NEAR, CONVERSIO:NEAR, PROCES:NEAR, CALCUL:NEAR,
VISUAL:NEAR
EXTRN       SORTIDA:NEAR, ERROR:NEAR

DADES  SEGMENT PARA PUBLIC  ;definicio de les variables del programa
          OPER1  DB      1              ; OPERAND 1 (BYTE)
          OPER2  DB      1              ; OPERAND 2 (BYTE)
          OPERACIO       DB     ?       ; operacio a realitzar (BYTE)
          RESULTAT       DW     1       ; RESULTAT (WORD)
          ENTRADAD       DB     6 DUP(0)     ; ENTRADA DADES
                                             ; (ARRAY OF BYTES)
          NBCD           DB     5 DUP(0)     ; resultat de la conversio a
                                             ;BCD desempaquetat
          ASBCD          DB     6 DUP(' ')   ; resultat de la codificacio
                                             ;BCD desempaquetat a ASCII
          RESPOSTA       DB     ?            ; RESPOSTA FINAL USUARI 0 -
NO, 1 SI
          TRL_SCR        DB     80 DUP(' '),'$'   ; NETEJA PANTALLA
          MISS_INDB      ' EPT - PRACTIQUES-96  -  PRACTICA DE LA
CALCULADORA',0AH,0DH,'$'
          MISS_ENT       DB     ' ENTRA: operand1  operacio  operand2 i
(=)'
                         DB    ' (CLEAR per esborrar): $'
          MISS_END       DB     ' VOLS ENTRAR MES VALORS ?  (0 - NO, 1 -
SI ): $'
          MISS_FIN       DB     ' EPT - PRACTIQUES-96  -  A REVEURE FINS
UNA ALTRA OCASIO'
                         DB     0AH,0DH,'$'  ; salt de linia del
missatge anterior
          MISS_LINE      DB     0AH,0DH,'$'       ; SALT DE LINIA (cr ,
lf)
          MISS_ERR1      DB     0AH,0DH,' ERROR: Masses entrades! Torna-
ho a provar!'
                         DB     0AH,0DH,'$'
          MISS_ERR2      DB     0AH,0DH,' ERROR: Format incorrecte!
Torna-ho a provar!'
                         DB     0AH,0DH,'$'
          MISS_ERR3      DB     0AH,0DH,' Esborrat! Torna-ho a
provar!',0AH,0DH,'$'
DADES  ENDS
```

```
PILA  SEGMENT  STACK
         DW  128  DUP ('S')          ;definicio de la pila del
programa
PILA  ENDS

CODI      SEGMENT  PARA PUBLIC
          ASSUME  CS:CODI, DS:DADES, ES:DADES, SS:PILA

PRINCI    PROC

ENTRADA: MOV  AX,DADES          ; punt d' entrada al
         MOV  DS, AX            ; programa i
         MOV  ES, AX            ; inicialitzacio del registres de
segment
         CALL BORPANT           ; instruccions del programa principal
         CALL BENVING           ; missatge de benvinguda
CONTINUAR:
         CALL DEMENTR   ; demanar l' entrada de dades
         MOV CX, 6              ;
         MOV SI, 0              ; SI fara d' index sobre la variable
ENTRADA
ENTRAOPER:
         CALL TECLA             ; demanar tecla
         CALL CONVERSIO         ; convertir a binari
         MOV ENTRADAD[SI], DL ; guardem el valor binari
         CALL VISUAL            ; visualitzar l' entrada
         CMP ENTRADAD[SI], 14 ; determinar si s'ha pitjat la tecla =
         JE TIGUAL
         CMP ENTRADAD[SI], 15  ; es la tecla CLEAR?
         JE TCLEAR
         INC SI
         LOOP ENTRAOPER         ; entrar com a maxim 6 bytes
         MOV AL, 1              ; error: masses entrades.
         CALL ERROR
         JMP NOVA_ENT           ; saltar al final
TCLEAR: MOV AL, 3               ; s' ha acabat per tecla CLEAR
         CALL ERROR
         JMP CONTINUAR
TIGUAL: CALL PROCES             ; analitzar l' entrada.
         CMP AL, 0FFh           ; es correcta
         JNE NOERROR            ; si no es correcta fer el calcul
         MOV AL, 2              ; error 2, treure missatge.
         CALL ERROR
         JMP NOVA_ENT           ; tornar a entrar
NOERROR:
         CALL CALCUL            ; realitzar el calcul si entrada
correcte
         CALL SORTIDA           ; treure el resultat
```

```
NOVA_ENT: CALL COMIAT           ;
          CMP RESPOSTA, 1       ; continuar amb una altra operacio?
          JE CONTINUAR          ; saltar si mes operacions.
          CALL ULTIMMIS         ; si no, missatge de comiat i
          MOV AX, 4C00H         ; tornar el control al sistema operatiu
          INT 21H               ; i acabar el programa
PRINCI    ENDP

DEMENTR   PROC            ; subrutina DEMENT per a l' entrada dels
operands i operacio
          MOV DX, OFFSET MISS_LINE     ; demanar entrada de operands.
          MOV AH, 9
          INT 21H
          MOV DX, OFFSET MISS_ENT      ; demanar entrada de operands.
          MOV AH, 9
          INT 21H
          RET
DEMENTR   ENDP

BORPANT   PROC
          ; subrutina BORPANT del programa per esborrar la pantalla
          PUSH CX
          MOV CX,25
BORLIN: MOV DX, OFFSET TRL_SCR ; visualitzar linies en blanc per
esborrar la pant.
          MOV AH, 9
          INT 21H
          LOOP BORLIN
          POP CX
          RET
BORPANT   ENDP

BENVING   PROC    ;subrutina BENVING del programa pantalla de benvinguda
          PUSH CX
          MOV CX, 10
SCROLL: MOV DX, OFFSET TRL_SCR ; scroll vertical
          MOV AH, 9
          INT 21H
          LOOP SCROLL
      MOV DX, OFFSET MISS_IN           ; missatge de benvinguda
          MOV AH, 9
          INT 21H
          MOV CX,15
SCROLLDE: MOV DX, OFFSET TRL_SCR       ; scroll vertical.
          MOV AH, 9
          INT 21H
          LOOP SCROLLDE
          POP CX
          RET
BENVING   ENDP
```

```
COMIAT    PROC                    ; subrutina BENVING del programa
          MOV DX, OFFSET MISS_LINE     ; salt de linia
          MOV AH, 9
          INT 21H
          MOV DX, OFFSET MISS_END      ; preguntar si es vol continuar
          MOV AH, 9
          INT 21H
TORNAE: CALL TECLA
          CALL CONVERSIO
          CMP DL, 0                ; comprovar si la resposta es correcta
          JE RETORNA
          CMP DL, 1
          JNE TORNAE               ; si no es ni 0 ni 1 tornar a demanar
RETORNA:
          MOV RESPOSTA, DL             ; guardar resposta i retornar
          MOV DX, OFFSET MISS_LINE     ; salt de linia
          MOV AH, 9
          INT 21H
          RET
COMIAT    ENDP

ULTIMMIS  PROC                     ; subrutina BENVING del programa
          PUSH CX
          MOV CX, 12
SCROLL1: MOV DX, OFFSET TRL_SCR
          MOV AH, 9
          INT 21H
          LOOP SCROLL1
      MOV DX, OFFSET MISS_FIN          ; missatge de comiat
          MOV AH, 9
          INT 21H
          MOV CX, 12
SCROLL2: MOV DX, OFFSET TRL_SCR
          MOV AH, 9
          INT 21H
          LOOP SCROLL2
          POP CX
          RET
ULTIMMIS  ENDP
CODI  ENDS

          END  PRINCI              ; final del modul font i

                                   ; indicacio punt entrada al programa
```

Subrutines principals, rutines. asm.

Codi font de les subrutines de la Pràctica 2 ***rutines . asm.***

Les parts en negrita son les que ha de desenvolupar l' alumne en aquesta pràctica.

```
TITLE    RUTINES      ;

PUBLIC   TECLA, CONVERSIO, PROCES, CALCUL, VISUAL, SORTIDA,
       ERROR

EXTRN    OPER1:BYTE, OPER2:BYTE, RESULTAT:WORD, ENTRADAD:BYTE,
       NBCD:BYTE, ASBCD:BYTE
EXTRN    MISS_ERR1:BYTE, MISS_ERR2:BYTE, MISS_ERR3:BYTE,
       MISS_LINE:BYTE, OPERACIO:BYTE

PROTI EQU 408H
RETARD EQU 65000

DADES    SEGMENT   PARA   PUBLIC
         PORTD DW ?
         PORTE DW ?
         PORTC DW ?
         VPORTC DB ?
         TAULA DB     31H, 20H, 21H, 22H, 10H, 11H, 12H, 00H,
01H, 02H, 03H, 13H, 23H, 33H
               DB     32H, 30H
         TSIGNES DB  '+','-','*','/','='
DADES    ENDS

CODI     SEGMENT  PARA  PUBLIC
         ASSUME   CS:CODI, DS:DADES, ES:DADES

TECLA    PROC                  ;  Rutina deteccio tecla pitjada
                               ;  cal salvar CX, DS i AX
                               ;  detecta  fila i columna
         .   .   .   .         ;  instruccions  rutina
         .   .   .   .
         .   .   .   .

         RET
TECLA    ENDP
```

```
CONVERSIO       PROC
        PUSH SI                     ; Rutina conversio tecla: fila i
columna
        PUSH CX                     ; a binari  0.. 15
        AND AL, 0Fh            ; posar les coordenades en unic byte
        MOV CL, 4
        SHL AH, CL
        OR AL, AH              ; en AL hi ha fila.columna en un
byte
        MOV SI, -1             ; preparar la conversio
        MOV CX, 16
SELEM:INC SI            ; buscar les coordenades a la taula
        CMP TAULA[SI], AL
        LOOPNE SELEM
        MOV DX, SI             ; la posicio de les coordenades en
la taula es el valor de conv.
        POP CX
        POP SI
        RET
CONVERSIO       ENDP

PROCES  PROC                   ; Rutina  proces
                        ; procesa dades entrada Num1, Num2 i oper
        .   .   .   .          ;  instruccions  rutina
        .   .   .   .
        .   .   .   .
        RET
PROCES  ENDP

CALCUL  PROC                   ;  Rutina  calcul:  efectua
                               ; Num1 oper Num2 = resultat
        .   .   .   .          ;  instruccions  rutina
        .   .   .   .
        .   .   .   .
        RET
CALCUL  ENDP

VISUAL  PROC                   ; visualitza un caracter
                               ; convertint-lo a ASCII i cridant
                               ; INT 21
        .   .   .   .          ;  instruccions  rutina
        .   .   .   .
        .   .   .   .
        RET
VISUAL  ENDP
```

```
SORTIDA PROC
                                        ; extraure resultats
                                        ; instruccions  rutina
                                        ; cal salvar  BX, SI i DX
        .   .   .   .           ;  instruccions  rutina
        .   .   .   .
        .   .   .   .

        RET
SORTIDA ENDP

ERROR   PROC
                                ; instruccions  rutina
        CMP AL, 1               ; determinar el tipus de missatge a
visualitzar
        JNE ERR2
        MOV DX, OFFSET MISS_ERR1
        JMP VISERR
ERR2: CMP AL, 2
        JNE ERR3
        MOV DX, OFFSET MISS_ERR2
        JMP VISERR
ERR3: MOV DX, OFFSET MISS_ERR3
VISERR:
        MOV AH, 9
        INT 21H
        RET
ERROR   ENDP

ESPERA20 PROC
        MOV CX, RETARD
BUCLE: LOOP BUCLE
        RET
ESPERA20 ENDP

CODI    ENDS
        END
```

Rutina retard 20 mseg.

```
; No té paràmetres d' entrada
; Condicions de sortida:
;    Modifica el registre CX
```

```
ESPERA20 PROC   ; rutina de retard de 20 ms
        MOV CX, RETARD
BUCLE:LOOP BUCLE
        RET
ESPERA20  ENDP
```

Nota: RETARD és un símbol definit amb un valor tal que permeti fer un retard de 20 ms segons la relació següent:

RETARD = (TEMPS x FR) / 17
FR = freqüència de rellotge del sistema (MHz)
TEMPS = temps de retard (µs)

Entrada i sortida DOS.

Per a fer les entrades sortides de dades, cal utilitzar les utilitats del sistema operatiu DOS, tal com es descriu a continuació:

Per treure un caràcter per pantalla:

Paràmetres d' entrada:
AH = 2 ; indica el tipus de servei. DL : ha de contenir el codi ASCII del caràcter que es vol visualitzar

Condicions de sortida: Modifica el valor de AL i treu per pantalla el caràcter donat en DL

```
MOV DL, caràcter ASCII
MOV AH, 2
INT 21H
```

Per treure una cadena de caràcters per pantalla:

Paràmetres d' entrada:
AH = 9 ; indica el tipus de servei
DX : ha de contenir el desplaçament de la cadena a visualitzar. Aquesta cadena ha d' estar acabada amb el caràcter '$', que serveix de delimitador.

Condicions de sortida:
Modifica AL

```
MOV DX, OFFSET cadena ASCII
MOV AH, 9
INT 21H
```

Annex 2.

Programació en ensamblador per a 8086

Programació en ensamblador per a 8086.

A l' hora de construir un programa que ha de ser executat per un microprocessador es pot fer servir el llenguatge de programació ensamblador. Aquest llenguatge ens permet tenir a l' abast tots els recursos i potència del microprocessador, a la vegada que ens ofereix una major facilitat, estructuració i inteligibilitat que no pas el llenguatge màquina (que consistiria en donar els codis de les instruccions, que formen un programa, en binari o hexadecimal).

Un programa en llenguatge ensamblador consisteix en un texte, programa font, que conté una sèrie de símbols i dades. Aquests símbols correspondran a mnemònics d' instruccions del microprocessador, directives, variables,... Per tal que aquest programa font pugui ser executable pel microprocessador s' ha de traduir a instruccions vàlides del microprocessador, o sigui, que s' han de generar els codis que corresponen a les instruccions donades en forma de menmònics i operands, i que el microporcessador és capaç d' executar.

Procés de creació d' un programa executable

1.- Un cop s' ha dissenyat el programa s' ha d' entrar a l' ordinador a través d' un editor de texte, i així obtindrem el programa o mòdul font. Aquest texte només pot estar format a base dels caràcters ASCII, sense codis de control particulars de cada processador de texte. Es podran fer servir, per exemple: EDLIN, NE, o un editor d' altres llenguatges de programació, per exemple l' editor del Turbo Pascal, etc.

2.- El texte del mòdul font s' ha de traduir a codis que el microprocessador pogui entendre. Per fer aquesta operació es fa servir un programa ensamblador. El resultat que ens dóna l' ensamblador a partir d' un programa font és el mòdul o programa objecte. En aquest mòdul tindrem tots els mnemònics traduïts a codis de instruccions de microprocessador, però quedaran adreces per assignar. Aquest últim fet permetrà que el programa pugui ser colocat en qualsevol posició de memòria però encara no serà directament executable.

Codificació de les instruccions

Els codis de les instruccions estan formats per diversos camps que determinen la instrucció que és, el mode d' adreçament, etc.:

En llenguatge màquina les instruccions del 8086 tenen un format general tal com el següent:

Codi operació	mod reg	r/m	desp

Codi és el codi d' operació de la instrucció. Ocupa el primer byte de la instrucció i sempre hi és. Dins de codi hi haurà dos camps més:

w = un bit que dóna la longitud dels operands: 0 - byte , 1 - paraula.

d = un bit que especifica qui és el destí:

0 - s' especifica amb mod i r/m

1 - s' especifica amb reg

Després del codi d' operació hi pot haver un camp que especifica el mode d' adreçament:

mod = 2 bits. Indiquen el tipus de desplaçament:

mod	desplaçament
00	0
01	8 bits en el següent byte (extensió de signe a paraula)
10	16 bits en els dos bytes següents.
11	r/m és un registre

r/m = 3 bits. Indiquen el tipus d' adreçament de l' operand.

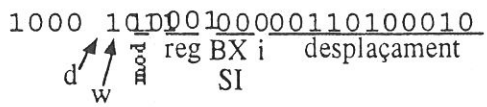
reg = 2 o 3 bits. Indica el registre que s' utilitzarà com operand.

des és un o dos bytes que contenen el desplaçament d' una posició de memòria.

val és un o dos bytes que contenen un valor inmediat.

Exemple : La instrucció MOV CX,COMPTADOR[BX][SI]:

on COMPTADOR = variable tipus paraula amb desplaçament 1234h



3.- Un cop obtingut el mòdul font, aquest seveix d' entrada al programa enllaçador ("linkador") que dóna com a resultat el programa executable. El programa enllaçador acaba de definir les adreces que en el mòdul objecte no havien estat concretades, de manera que el programa ja estarà a punt de posar-se en la posició que el sistema operatiu li assigni.

El programa enllaçador pot tenir com entrada diversos mòduls objecte. En aquest cas, s' enllaçarà de manera que obtindrem un únic programa executable. Així, es poden aprofitar mòduls objecte que contenen rutines dissenyades per a altres programes però

que interessen en el nostre, permetent estalviant feina de programació o repartir la mateixa tasca entre vàries persones.

També es poden crear fitxers anomenats llibreries (amb extenció .LIB) que contenen diversos mòduls objecte amb alguna característica comú. Per exemple, podriem tenir una llibreria de rutines de tractament matemàtic, una altra de rutines de tractament de fitxers, etc. De manera que en els nostres programes podriem fer crides a la subrutina desitjada sense haver de programar-la de nou, i només caldrà declarar, prèviament, que es tracta d' una rutina definida externament.

Després, durant el procés d' enllaçat, indicaríem a l' enllaçador les llibreries que ha d' utilitzar perquè incorpori les rutines cridades al fitxer executable final.

Tot seguit donarem una referència, encara que breu, als elements que contitueixen el llenguatge ensamblador, basant-nos en el Turbo Assembler (TASM). Aquí només pretenem fer un primer contacte amb el llenguatge que serveixi per donar una visió global i ràpida, però que, al mateix temps, permeti tenir les bases mínimes per poder fer programes en llenguatge ensamblador.

Contingut del mòdul font

Un programa en ensamblador estarà constituït per línies de sentència. Les sentències poden ser:

- Una instrucció de llenguatge ensamblador.
- Una directiva de l' ensamblador.

Les sentències també inclouen operands que tenen la missió de facilitar la programació i d' evitar possibles ambigüitats.

En una sentència es poden emprar contants del següent tipus:

Binària: seqüència d'1 i 0 amb una B al final.

Exemple: 01000111B és 71 decimal

Decimal: seqüència de dígits decimals (0 al 9) amb o sense una D al final.

Exemple: 84d o simplement 84

Hexadecimal: seqüència de dígits (0 al 9) o lletres (A a la F) amb una H al final.

Exemple: 0FFh és el 255 decimal

S' ha de posar un 0 davant d' una que comenci per una lletra (A a F).

Caràcter: Una cadena de lletres, dígits o símbols alfanumèrics entre " " o ' '.

Exemple: 'CADENA CARACTERS'

Números negatius: Decimal, amb un - al davant

Binari o hexadecimal, en complement a dos.

Instruccions de llenguatge ensamblador

Ocupen una línia i poden tenir quatre camps:

```
[etiqueta:] mnemònic [operands] [;comentari]
```

Camp mnemònic: expressa un mnemònic del 8086 i és l'únic camp obligatori.

Camp etiqueta: dóna un nom a la posició que ocupa una instrucció. Així podem fer referència a la instrucció a través de l' etiqueta. Pot tenir 31 caràcters (A...Z, a...z, 0..9 i ? . @ _ $) i no pot començar per un número. Si es fa servir el punt ha de ser el primer caràcter de l' etiqueta. No es pot fer servir qualsevol símbol que ja estigui predefinit.

Camp operand: dóna els operands que ha de fer servir i com els ha d' obtenir.

Per exemple: MOV CX,DX

l' operand font és el valor del registre DX i l' operand destí és el registre CX.

Camp comentari: va precedit de (;) i permet inserir en la línia de sentència un comentari referent a la funció que fa una instrucció, una rutina, o un programa, etc. Els comentaris són un bon costum si es vol que un programa sigui de fàcil entendre pel programador. La documentació d' una programa és molt important encara que sigui un programa que només l' hagi de tractar la persona que l' ha fet.

Exemple de línies d' instruccions de llenguatge ensamblador:

```
        XOR AX,AX                       ; esborrar AX
        MOV AX,SEGMENT_FONT     ; inicialització de registres
de segment
        MOV DS, AX              ;
        MOV AX,SEGMENT_DESTI    ;
        MOV ES,AX       ;
```

Si es vol que una línia només sigui de comentaris el punt i coma (;) ha de ser el primer caràcter.

Directives d' ensamblador

Les directives controlen, donen informació a l' ensamblador i determinen algunes característiques del codi objete que generarà; però, a diferència de les instruccions de llenguatge ensamblador, no genera codi objecte. Les veurem agrupades segons la funció que realitzen.

Directives de definició de símbols

EQU nom **EQU** expressió o texte

Assigna el *nom* simbòlic a *expressió o texte*. *Expressió* pot ser una constant de 16 bits, una referència a memòria, un altre nom simbòlic, un identificador de segment (prefixe), un operand o una etiqueta d' instrucció. Després, *nom*, es pot emprar en qualsevol lloc on es faria servir normalment l' *expressió o texte*.

Els símbols definits amb EQU no es poden redefinir. Exemples:

```
C          EQU  100
PUNTER     EQU  DS:[BP][SI]        ; ara PUNTER equival a
DS:[BP][SI]
COMPTA     EQU   CX  ; posar COMPTA és el mateix que posar CX
```

= nom = expressió numèrica

Assigna el valor de l' *expressió numèrica* al *nom*. Funciona de manera semblant que EQU però la directiva = permet tornar a canviar el valor assignat a *nom* i el valor de l' expressió no pot ser texte, sinó numèric:

```
consta = 30h
consta = consta -2  ; redefinició de consta
```

Directives de definició de dades

En un programa, és habitual que es guardin valors de dades a la memòria. Per poder fer referència a aquests valors d' una manera més còmoda se'ls hi dóna un nom (símbol o nom de variable). Aquesta etiqueta representa l' adreça on està el valor. Degut a que el 8086 és capaç de processar diverses longituds d' operands, per tal que l' ensamblador pugui saber quin codi d' instrucció ha de generar a partir de la instrucció font, cal definir la longitud de les dades a les que fan referència els símbols. Per aconseguir això hi ha 5 directives que tenen el següent format:

```
nom   DB   expressió[,...]   ; definició de byte
nom   DW   expressió[,...]   ; definició de paraula
nom   DD   expressió[,...]   ; definició de doble paraula
nom   DQ   expressió[,...]   ; definició de quadruple paraula
nom   DT   expressió[,...]   ; definició de deu bytes
```

nom és un símbol i *expressió* és un valor d' inicialització. Els símbol [,...] volen dir que es poden incloure vàries expressions separades per comes.

Aquestes directives també es poden interpretar com una manera de reservar memòria en el moment d' ensamblar, amb la possibilitat de donar un valor en el moment de fer la reserva, mitjançant el camp expressió.

Exemple:

```
consta   EQU 4
VALOR_MAX      DB        255
TAULA_WORD     DW        9,45,30*6,consta-4
```

Això equival a tenir una zona de memòria amb la següent estructura:

0	FFh	VALOR_MAX
1	09	TAULA_WORD
2	00	
3	2Dh	
4	00h	
5	B4h	
6	00	
7	00	
8	00	

L' ensamblador té un operador que permet inicialitzar un zona de dades amb valors iguals sense haver d' escriure'ls un a un: **DUP**

```
TAULA_1   DB   20 DUP(0)  ;reserva 20 bytes i a
                          ;cadascun li dóna valor 0
```

Les variables definides amb les directives anteriors tenen uns atributs que l' ensamblador calcula i guarda:

SEG : segment on s' ha definit la variable.

OFFSET : desplaçament de la variable dins del segment

TYPE : nombre de bytes dels elements d' una variable.

Tipus:	valor de l' atribut:
BYTE	1
WORD	2
DWORD	4
NEAR	0FFFFh
FAR	0FFFEh
Constant	0
Estructura	Nombre de bytes de l' estructura.

LENGTH: Conté un valor que correspon al nombre d' elements de la variable, sempre que fos reservada fent servir l' operador DUP. En cas contrari sempre té el valor 1.

SIZE : nombre total de bytes d' una variable:

TYPE*LENGTH

Aquests noms d' atributs es poden fer servir també com operadors per obtenir el valor corresponent d' atribut:

```
taula  DW 100 DUP(2233H)
N_paraul EQU  LENGTH taula        ; N_Paraul
indicad  DB   0
MOV DX,OFFSET indicad        ; DX rep el valor de
                           ;desplaçament de la
                               ; variable indicad.
```

També hi ha un operador que permet contrarestar l' atribut de tipus d' una variable; el format és :

tipus PTR expressió

Exemples:

```
taula_2   DW    50 DUP(0)
Primerb EQU  BYTE PTR taula_2   ; Primerb apunta al
                     ;primer byte de taula_2
MOV WORD PTR [BX], 5            ; indica que BX és un
                                ;punter a paraula
                     ; per tant 5 és interpretat
                     ;com un valor de dos bytes.
ENTRADA: MOV AX, 0
         ...
      JMP  FAR PTR ENTRADA     ; es fa un salt llarg.
```

Hi han dues directives de definició de dades més que permeten definir estructures de dades més complicades a partir de les bàsiques que s' han tractat anteriorment, facilitant l' accés a cadascun dels elements. En aquest sentit, les directives **STRUC** i **ENDS** defineixen una plantilla de camps a nivell de bytes que es farà servir per reservar memòria, caracteritzant-la amb una certa estructura.

La sintaxi de definició és la següent:

nom_estructura **STRUC**

...

[nom_variable] directiva_definició_dades expressió ; camp

...

nom_estructura **ENDS**

nom_estructura és el símbol que representa a l' estructura de dades definit. *nom_variable* serveix per donar un nom a cada camp. Un camp pot ser simple (d' un sol element) o múltiple.

Un cop s' ha definit l' estructura, la reserva de memòria i inicialitació dels camps es farà de la manera següent:

[nom] nom_estructura<[exp][,...]>

o bé

[nom] nom_estructura expressió DUP(<[exp]>)

nom és una etiqueta que representa al primer byte de la zona reservada i *exp* és un valor de modificació del camp corresponent. Un camp es pot modificar en la reserva només si és simple.

Per fer referència a un dels camps en un programa es fa:

variable.camp

Exemple:

```
ESTRUCTURA    STRUC
    CAMP1      DB  2,3   ; camp múltiple
    CAMP2      DB 20 DUP(?)   ; camp múltiple
    CAMP3      DW  10    ;camp simple
    CAMP4      DB  ' aBC'       ;camp simple
ESTRUCTURA    ENDS
```

Per reserva una zona de memòria amb aquesta estructura de dades:

```
ZONA    ESTRUCTURA<,,7,'DEF'>
```

Per fer-hi referència:

```
MOV AL,[BX].CAMP1          o      MOV AL,ZONA.CAMP1
(Pel primer cas BX ha de contenir el desplaçament
de ZONA)
```

Directives d' especificació de procediment

Les directives d' especificació de procediment donen un nom a un grup d' instruccions de manera que es podrà fer referència a aquestes instruccions a través del nom del procediment. Si el grup d' instruccions acaba amb una instrucció de retorn (per exemple RET) llavors, aquest grup, es comportarà com una subrutina. El format d' aquestes directives és el següent :

```
nom_procediment  PROC [atribut]
...
      grup d' instruccions
...
      RET ; si volem que es comporti com una subrutina
nom_procediment ENDP
```

nom_procediment és una etiqueta que senyala a la posició que ocupa la primera instrucció del grup.

Amb *atribut* es defineix l' atribut de distància del procediment. Pot ser NEAR (valor per defecte) o FAR. Els procediments amb atribut NEAR només poden ser cridats des del segment on estan definits.

El procediment principal ha de tenir atribut FAR.

Exemple:

```
SUMAR PROC NEAR      ;Suma de quadrats OPE1 OPE2
   PUSH BX           ; salvar BX
   XOR AX,AX         ; AX=0
   MOV AL,OPE1       ; obtenir 1er operand
   MUL OPE1          ; operand al quadrat
   MOV BX,AX         ; salvar 1er quardrat
   XOR AX,AX                ;
   MOV AL,OPE2       ; segon operand
   MUL OPE2          ; 2on al quadrat
   ADD AX,BX         ; sumar quadrats
   POP BX
   RET
SUMAR ENDP
```

Directives d' especificació de segment

Aquestes directives divideixen el programa font en segments. Com ja sabem, el 8086 pot treballar sobre quatre segment de memòria:

- Segment de dades, adreçat a través del registre de segment DS
- Segment de codi, adreçat a través del registre de segment CS
- Segment de pila, adreçat a través del registre de segment SS
- Segment alternatiu de dades, adreçat a través de ES

En l' especificació es fan servir les directives **SEGMENT** i **ENDS** amb el format següent:

```
nom_seg SEGMENT [tipus-alin] [tipus-combi] ['classe']
. . .
nom_seg ENDS
```

El paràmetre *tipus-alin* (tipus d' alineació) indica l' adreça a partir de la qual començarà el segment. Pot ser qualsevol (tipus-alin = BYTE), en adreça parella (WORD), en adreça múltiple de 16 (PARA) o múltiple de 256 (PAGE).

El paràmetre *tipus-combi* (tipus de combinació) especifica la forma en que es combinaran els segments que tinguin el mateix mon. Els segments de codi, de dades i dades alternatives es poden unir (tipus-combi = PUBLIC) o es poden solapar (COMMON). Els segments de la pila han de ser del tipus STACK.

El paràmetre classe afecta a l' ordre en que s' emmagatzemen els segments; els segments que tenen el mateix nom s' emmagatzemen consecutivament i els de nom diferent en l' ordre que els troba el programa enllaçador.

Normalment només es fan servir els tipus PARA i PUBLIC.

Per associar un segment a un registre de segment es fa servir la directiva **ASSUME**:

```
ASSUME reg.seg : nom_seg [,...]
```

on reg.seg pot ser qualsevol registre de segment: CS, DS, ES, SS.

Aquesta directiva permet saber a l' ensamblador si ha de posar un prefixe de canvi segment davant d' una instrucció amb accés a memòria.

ASSUME no assigna cap valor als registres de segment. Aquesta directiva s' ha d' indicar inmediatament després de SEGMENT. Per cancel.lar la relació d' un o de tots els registres de segment amb segments especificats es fa servir l' operador *NOTHING*:

```
ASSUME ES:NOTHING ; anul.la les relacions fetes a ES
ASSUME NOTHING    ; anul.la les relacions de tots els
registres de segment.
```

També es pot indicar un canvi de segment posant-ho explícitament en la instrucció amb DS:, ES:, SS:, o CS:

Exemples:

```
DADES SEGMENT PARA PUBLIC 'dades'
    ASSUME ES:DADES

    DADA  DB  1

DADES ENDS
```

```
CSEG SEGMENT PARA PUBLIC 'codi'
    ASSUME CS:CSEG
    MOV DH,DADA                ; s' entén ES com a reg. de
                               ;segment
    MOV DH,BYTE PTR CS:[BX]        ; fa servir el
                                   ;registre CS
CSEG ENDS
```

En aquest exemple, l' ensamblador considera que totes les variables que es defineixin entre ASSUME i ENDS del segment DADES tenen la seva adreça de segment en el registre ES.

Directives de control de l' ensamblador

END *END [expressió]*

Indica el final del programa font. *expressió* indica l' adreça de començament del programa i normalment s' especifica amb una etiqueta.

Si el programa consta d' un sol mòdul font, expressió és opcional. Si hi ha diversos mòduls que s' han d' enllaçar junts, *expressió* només s' especifica en el mòdul principal. Si el mòdul principal no el fa servir llavors no es passa adreça de començament de programa a l' enllaçador.

ORG *ORG expressió*

Posa el comptador de posició de l' ensamblador al valor d' *expressió*. L' ensamblador guardarà el codi següent a partir d' aquesta adreça (desplaçament dins del segment actual).

Si *expressió* és $ manté el valor actual del comptador. ORG no es pot emprar dins de la definició d' una estructura.

Exemple:

```
ORG  100H  ; la nova posició serà el desplaçament 100h
ORG  $+16  ; la nova posició serà el desplaçament actual
           ; més 16
```

EVEN

Força al comptador de posició a un valor parell, introduint una expressió NOP (1 byte) en la posició actual del programa si el comptador de posicions tenia un valor senar.

Altres directives

.RADIX *.RADIX expressió*

Serveix per canviar la base d e numeració per defecte de l' ensamblador, a la donada per *expressió*, que s' interpreta sempre en base 10. Per defecte els valors numèrics sense subfixe es consideren en base 10. Els subfixes són:

b - binari

d - decimal

o/q - octal

h - hexadecimal

.Radix no afecta a DD, DQ o DT on sempre es suposen valors en decimal, a no ser que s' especifiqui un subfixe de tipus de dada. Quan es dóna .RADIX 16 s' ha d' anar en compte amb els valors acabats amb una D ja que l' ensamblador els pot considerar en base 10. En aquest cas s' ha de donar el subfixe H:

```
.RADIX 16
    DB 11D     ; sempre és 11 en base 10
    DB 11DH     ; sempre és 11D en base 16
```

TITLE *TITLE texte*

texte apareix en la primera línia de cada pàgina d' un llistat (màxim 60 caràcters). Només és posa en un dels mòduls que formin el programa. Els primers 6 caràcters es fan servir com a nom del mòdul objecte, sempre que no s' especifiqui un altre amb la directiva NAME. Si no hi ha ni TITLE ni NAME el nom del mòdul OBJ és el del mòdul ASM.

NAME *NAME nom_mòdul_objecte*

Dóna un nom al mòdul objecte que genera l' ensamblador. No pot ser una paraula reservada. Només es pot especifica un NAME per mòdul.

LABEL *nom LABEL tipus*

Permet assignar diferents noms a variables o etiquetes amb possibilitat de variar el seu tipus. *nom* pot ser:

- El nom d' una estructura.
- Una etiqueta.

tipus pot ser:

per a zona de dades:

BYTE
WORD
DWORD
nom_estructura
nom_registre

per a codi executable:

NEAR (mateix segment)
FAR (un altre segment)

No es pot fer servir dins d' una definició d' estructura.

Exemple:

```
TAULA DW 0        ;paraula amb valor 0
TAULAB LABEL BYTE
TAULA DW 0        ; TAULAB i TAULA fan referència a la
                  ; mateixa adreça però amb diferents atributs
```

Llavors podriem fer:

```
MOV AL,TAULAB          ;mou un byte
MOV AX,TAULA           ;mou una paraula
```

Exemple:

```
ETIQF   LABEL FAR
ETIQN:
```

ETIQF i ETIQN són etiquetes de la mateixa posició però amb atribut FAR i NEAR, respectivament.

PUBLIC *PUBLIC símbol[,...]*

El *símbol* definit podrà ser emprat per altres mòduls que han de ser enllaçats amb aquest.

EXTRN *EXTRN nom:tipus[,...]*

Especifica que *nom* és un símbol que ha estat definit en altres mòduls.

tipus especifica el tipus del símbol:

BYTE, WORD o DWORD si nom és símbol definit en un segment de dades.
NEAR o FAR si nom és etiqueta de procediment.
ABS si nom és una constant definida per EQU o =.

INCLUDE *INCLUDE nom_fitxer*

Afegeix el contingut del fitxer font especificat amb *nom_fitxer* al fitxer font actual a partir del lloc que ocupa la directiva INCLUDE i substituint a aquesta.

Macros

Les macros es poden considerar com directives definides pel programador. Consisteixen en una seqüència de sentències d' ensamblador. Aquesta seqüència té associat un nom a través del qual es fa referència a la seqüència. Així, allà on hi hagi d' anar la seqüència d' instruccions es posa el nom de la macro que la representa. L' ensamblador substitueix el nom de la macro per la seqüència de sentències que representa (no confongeu macros amb subrutines).

Les macros faciliten la feina del programador perquè estalvia haver d' escriure vàries vegades un mateix grup de sentències, a la vegada que es poden fer servir en altres programes. Faciliten la depuració de programes al poder depurar les macros de forma individual.

Definició d' una macro:

```
nom_macro MACRO [llista de paràmetres]
... sentències
ENDM
```

Parts de la definició d' una macro:

1. Encapaçalament. La directiva MACRO amb el *nom* davant

   La *llista de parametres* seran un conjunt opcional de "valors" d' entrada que poden canviar la funció de la macro.

2. Cos de sentències que defineix la macro.
3. ENDM delimitador de la macro

Exemples: Per sumar dades de longitud de paraula

```
SUMA_PAR MACRO OP2,OP1,SUM
MOV AX,OP1
ADD AX,OP2
MOV SUM,AX
ENDM
```

Per cridar o fer referència a una macro:

```
SUMA_PAR  DADA,BX,RESUL
```

Això equivaldria a haver posat:

```
MOV  AX,BX
ADD AX,DADA
MOV RESUL,AX
```

O si haguéssim fet:

```
SUMA_PAR BX,CX,DI
```

seria el mateix que haver escrit:

```
MOV  AX,CX
ADD AX,BX
MOV DI,AX
```

Operadors

Són modificadors de les directives i de símbols. N' hi ha de diversos tipus:

Aritmètics: +, -, *, /, MOD, SHR, SHL

Lògics: AND, OR, XOR, NOT

De relació: EQ, NE, LT, GT, LE, GE

De devolució de valors: $, SEG, OFFSET, LENGTH, TYPE, SIZE

D' atribut: PTR, CS:, DS:, ES:, SS:, SHORT, THIS, HIGH, LOW.

Models de programes .EXE i .COM

Quan es tracta de fer programes en ensamblador que s' han d' executar a través del sistema operatiu MS-DOS, aquests programes han de tenir unes determinades característiques.

Es poden generar dos tipus de programes (o fitxers executables):

- Fitxers de tipus .EXE
- Fitxers de tipus .COM

Fitxer executable de tipus .EXE

No té restriccions de longitud i ha de tenir definits, al menys, un segment de pila i un segment de codi. En el programa principal s' han d' incloure unes instruccions que permetran al programa retornar al MS-DOS quan hagi acabat la seva execució.

Model de programa principal

```
PAGE ,132              ; longitud de la pàgina del llistat
TITLE títol del programa
sentències EXTRN si s' escau. ; per declarà variables o
símbols externs
DSEG SEGMENT PARA PUBLIC 'DADES'
            Definició de dades
DSEG ENDS
PILA SEGMENT PARA STACK 'PILA'
      DB  100h DUP('PILA')
PILA ENDS

CSEG SEGMENT PARA PUBLIC 'CODI'
      ASSUME CS:CSEG,DS:DSEG,SS:PILA
ENTRADA PROC FAR       ; punt d' entrada
      PUSH DS          ; per guardar l' adreça de segement de
la PSP

      MOV AX,DSEG      ;inicialitza registres de segment
      MOV DS,AX        ;
      ...
         Instruccions del programa  principal

      MOV AX,4C00H              ; retorn al MS-DOS
      INT 21H
ENTRADA ENDP

      Altres possibles procediments

CSEG  ENDS
      END ENTRADA    ; fi de programa fon i indicació de punt
d' entrada
Per a un mòdul secundari:
PAGE ,80
TITLE títol del mòdul
PUBLIC NOMPROCE      ; declara les variables i símbols que
poden ser
(posar PUBLIC per a variables de dades, si s' escau),     ;
cridats des d' altres mòduls
DSEG SEGMENT PARA PUBLIC 'DADES'

 definició de dades

DSEG ENDS
CSEG SEGMENT PARA PUBLIC 'CODI'
      ASSUME CS:CSEG,DS:DADES
```

```
NOMPROCE PROC NEAR
        instruccions
        RET     ;retorn al programa que l' ha cridat

NOMPROCE ENDP
        altres procediments
CSEG ENDS
        END
```

Noteu que la directiva EXTRN del programa principal ha de ser:

```
            EXTRN NOMPROCE:NEAR
```

Fitxer executable de tipus .COM

La seva longitud està limitada a 64KB. Es defineix un únic segment i serà el mateix tant per les dades, pila i codi. Per tant, no es permet accedir a altres segments de dades, pila o codi. La reserva de memòria per a dades o definició de dades es farà dins d' aquests únic segment, abans del cos d' instruccions i després de la instrucció que hi ha d' haver després de la directiva ORG 100H.

Tots els procediments han de ser NEAR. L' eND final ha d' anar seguit de l' etiqueta de la primera instrucció.

Model de programa principal:

```
PAGE ,132
TITLE   títul del programa
        Directives EXTRN si s' escau

CSEG SEGMENT PARA PUBLIC 'CODI'
        ASSUME CS:CSEG,DS:CSEG,SS:CSEG
        ORG 100H

ENTRADA JMP COMEN

        Definició de dades

COMEN PROC NEAR

        Instruccions

        MOV AX,4C00H        ; retorn al MS-DOS
        INT 21H
COMEN ENDP

        definició d' altres procediments
```

```
CSEG   ENDS
       END   ENTRADA
```

Aplicacions a la programació en ensamblador

En el moment de posar-se davant del paper per dissenyar un programa s' ha tenir la intenció de realitzar una solució al problema de tipus modular amb l' aplicació de tècniques Top-Down. Una estructuració modular es basa en la divisió del problema (el programa a dissenyar) en parts senzilles i el més independent possible, i aplicant altra cop aquest procés sobre cadascuna de les parts fins que obtinguem una definició de cada part el suficientment senzilla com per poguer-la implementar directament sense complicacions.

A nivell de programació, aquesta concepció modular comporta la utilització de subrutines (o procediments) per realitzar cadascuna de les part en que haurem dividit el programa. D' aquestes subrutines, n' hi haurà que realitzaran la seva feina sense necessitat de dades o paràmetres d' entrada o de sortida (dades inicials i resultats) i per tant només caldrà definir el conjunt d' instruccions que descriu les operacions que ha de fer la subrutina.

Per altra banda, hi haurà subrutines que, per fer les feines pròpies, necessitaran unes dades inicials sobre les quals faran el procés i potser també produiran un resultat que necessitarà altres parts del programa. En aquest cas, a part de definir les operacions que ha de realitzar la subrutina s' haurà d' establir un conveni per tal que diferents parts es puguin passar informació, o sigui, perquè es puguin comunicar.

Per realitzar aquesta comunicació entre procediments es podran emprar diversos mètodes que ara descriurem en termes generals i amb aplicació a la programació en ensamblador.

Comunicació a través de la memòria mitjançant variables compartides.

Es definiran varibles on els procediments posaran les dades a passar. Aquestes variables, doncs, hauran de ser conegudes i accessibles pels procediments implicats.

Per exemple: si definim un procediment que ha de realitzar la suma de valors de doble paraula, la transferència dels valors a sumar i del resultat es podria convenir que es fes a través de les variables DP1, DP2 i SD. Llavors :

```
DADES SEGMENT
      DP1   DD   10H   ; variables per al pas de paràmetres
      DP2   DD   20H
      SD   DD   ?      ; ...     altres dades del programa
DADES   ENDS
```

```
SUMAD  PROC     ;Suma dues paraules. DP1 i DP2 operands
       PUSH AX  ; SD resultat
       MOV AX,WORD PTR DP1     ; suma la primera paraula
       ADD AX,WORD PTR DP2
       MOV WORD PTR SD,AX
       MOV AX,WORD PTR DP1[2]          ; suma la segona paraula
       ADC AX,WORD PTR DP2[2]
       MOV WORD PTR SD[2],AX
       POP AX
       RET
SUMAD ENDP
```

Si el procediment està definit en el mateix mòdul des d' on és critat la definició de dades formarà part del conjunt de definicions de variables de la resta del programa i el procediments que vulguin els serveis del procediment SUMAD hauran de transferir les dades a les variables DP1 i DP2, després cridaran a la subrutina amb la instrucció CALL i finalment accediran a la variable SD per recuperar el resultat.

Si la definició del procediment es fa en un altre mòdul, es podrà fer referència directament a aquestes variables si en el mòdul que hi ha la definició de dades de les variables hi ha la directiva :

```
PUBLIC DP1, DP2, SD
```

i en el mòdul on no hi ha definició dades i en que són referenciades:

```
EXTRN DP1:DWORD, DP2:DWORD, SD:DWORD
```

Llavors, per obtenir el programa executable s' hauria d' enllaçar els mòduls de la següent manera:

TLINK PRINCIP+RUTINA

Una alternativa a aquesta última situació seria definir una àrea comú de dades per als dos mòduls, posant-la en els dos mòduls:

```
DADES SEGMENT COMMON
      DP1  DD  ?
      DP2  DD  ?
      SD  DD   ?
      ... ; altres definicions de dades
DADES ENDS
```

Per aquest cas l'única condició és que en la definició de dades de dada mòdul les variables DP1, DP2 i SD ocupin la mateixa posició respecta al començament del segment DADES. Per altra banda, no cal que aquestes variables tinguin el mateix nom l' altre mòdul.

Compartir variables entre procediments per fer el pas de paràmetres té el desavantatge que només es poden operar valors que hi hagi en aquestes variables. Per tant, cada cop que es vulgui aplicar el procediment sobre valors diferents s' hauran de carregar prèviament les dades en les variables que serveixen d' enllaç i després recollir el resultat. Aquest tipus de comunicació entre procediments és adequat quan hi ha poca quantitat d' informació a passar. En canvi, si les dades que s' han de transferir són moltes o d' estructura molt complicada el programa es fa més feixuc i la seva execució s' enlenteix. Moltes vegades es donarà la situació que s' han de passar pocs paràmetres i d' estructura molt senzilla, per exemple bytes o paraules, i es podrà optimitzar la comunicació transferint les dades a través de registres del microprocessador, fet que estalvia memòria i fa més ràpida l' execució ja que l' accés als paràmetres és més ràpid.

Comunicació passant l' adreça dels paràmetres.

Per evitar els inconvenients de passar els paràmetres a través de variables compartides, assenyalts anteriorment, es podria passar només l' adreça de la zona de memòria on es troben aquests paràmetres, en comptes de passar el valor. Així podrem indicar dades diferents cada vegada que es cridi al procediment sense haver de passar els valors explícitament. Estrictament parlant, a les variables de qui es passen les adreces s' anomen paràmetres.

Quan el nombre d' adreces de paràmetres a passar és petit, 2 o 3 com a màxim, es poden comunicar a través de registres de la CPU, procediment que permet estalviar memòria i que la transferència sigui ràpida.

Per salvar la limitació de quantitat que presenta el pas de paràmetres a través de registres, es podrien transferir a través de la pila. Mostrarem aquest procediment modificant l' exemple anterior:

```
PILA SEGMENT  STACK ; definició del segment de pila
      DB 100 DUP(?)
PILA ENDS
...
CODI  SEGMENT
      ASSUME ...
      MOV AX, DADES
      ...
      MOV AX, OFFSET DP1      ; guardar adreces a la pila
      PUSH AX
      LEA DI, DP2
      PUSH DI
      LEA DI,SD
      PUSH DI
      CALL SUMAD
      ...
SUMAD  PROC
      PUSH BP   ; salvar BP
      MOV BP,SP; obtenir punter de pila
```

```
        ADD BP,4        ; ajustar BP per apuntar a adreces de
paràmetres
        MOV DX,[BP]     ;obtenir adreces de paràmetres des de la
pila
        MOV DI,[BP+2]
        MOV SI,[BP+4]
        MOV AX,[SI]      ; fer la suma
        ADD AX, [DI]
        XCHG DX,SI       ;Obtenir adreça del resultat
        MOV [SI],AX      ; guardar part baixa del resultat
        XCHG DX,SI       ; recuperar adreça 1er operand
        MOV AX,[SI+2]    ; fer la suma de la part alta
        ADC AX, [DI+2]   ; sumar amb carry
        XCHG DX,SI       ;Obtenir adreça del resultat
        MOV [SI+2],AX    ; guardar part baixa del resultat
        POP BP
        RET   6         ; retornar restaurant el punter de pila
SUMAD ENDP
```

La instrucció RET 6 del procediment fa que SP es restauri al valor que tenia abans que el programa principal passés els paràmetres a la pila, de forma que SP apuntarà a la següent adreça de retorn, si és que n' hi havia.

Una altra forma de passar les adreces dels paràmetres és construint un taula d' adreces en la memòria i transferir l' adreça de la taula al procediment a través d' un registres o de la pila. Amb aquest mecanisme es podran passar gran quantitat d' informació encara que tingui una estructura complicada. Si tornem a utilitzar l' exemple per veure el funcionament d' aquest últim mètode tindrem:

```
PILA SEGMENT  STACK
      DB 100 DUP(?)
PILA ENDS
DADES SEGMENT
      DP1  DD  ?      ; variables
      DP2  DD  ?
      SD  DD  ?
TAULA_PARAMETRES  DW  3 DUP(?)       ;taula d' adreces
      ...
DADES  ENDS
CODI  SEGMENT
      ASSUME DS:DADES, ES:DADES, CS:CODI, SS:PILA
      MOV AX,DADES
      MOV DS,AX
      MOV ES,AX
      MOV AX, PILA
      MOV SS,AX
      ...
MOV TAULA_PARAMETRES, OFFSET DP1     ; passar paràmetres
      MOV TAULA_PARAMETRES+2, OFFSET DP2 ; a la taula
      MOV TAULA_PARAMETRES+4, OFFSET SD
      LEA BX, TAULA_PARAMETRES ; passem adreça de taula
```

```
        CALL SUMAD
        ...

SUMAD PROC
        MOV SI,[BX]     ; recuperar adeça de paràmetres
        MOV DI,[BX][2] ;...
        MOV DX,[BX+4]
        MOV AX, [SI]
        ADD AX, [DI]
        XCHG DX,SI
        MOV [SI],AX
        XCHG DX,SI
        MOV AX,[SI+2]   ; sumar la part alta
        ADC AX,[DI+2]
        XCHG DX,SI
        MOV [SI+2],AX
        RET
SUMAD  ENDP
```

Com sempre, la subrutina haurà de salvar, abans de fer res més, i recuperar, abans de retornar, el contingut dels registres que es vulguin mantenir després de l' execució del procediment cridat.

En general, a l' hora de fer el pas de paràmetres, es podran aplicar una combinació d' aquests mètodes. Molts cops interessarà escollir el mètode, o combinació de mètodes, que faci més eficient el programa, o sigui, que ens permeti obtenir un programa, ràpid, senzill i estructurat. Però sovint, i sobre tot, quan es tracta que un equip de persones facin un conjunt de rutines, per separat, que s' han de relacionar i cridar-se les unes a les altres, s' haurà de sacrificar part d' aquesta eficiència per fer un conveni de transferència de paràmetres clar i adient per a tots. Això també es donarà si es vol estandaritzar un grup de rutines sota un conjunt determinat de normes.Programació en ensamblador

A l' hora de construir un programa que ha de ser executat per un microprocessador es pot fer servir el llenguatge de programació ensamblador. Aquest llenguatge ens permet tenir a l' abast tots els recursos i potència del microprecessador, a la vegada que ens ofereix una major facilitat, estructuració i inteligibilitat que no pas el llenguatge màquina (que consistiria en donar els codis de les instruccions, que formen un programa, en binari o hexadecimal).

Un programa en llenguatge *ensamblador* consisteix en un texte, programa font, que conté una sèrie de símbols i dades. Aquests símbols correspondran a mnemònics d' instruccions del microprocessador, directives, variables,... Per tal que aquest programa font pugui ser executable pel microprocessador s' ha de traduir a instruccions vàlides del microprocessador, o sigui, que s' han de generar els codis que corresponen a les instruccions donades en forma de menmònics i operands i que el microporcessador és capaç d' executar.

Annex 3.

Resum instruccions ensamblador.

AAA	AAA (no operands) ASCII adjust for addition			Flags O D I T S Z A P C U U U X U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	AAA

AAD	AAD (no operands) ASCII adjust for division			Flags O D I T S Z A P C U X X U X U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	60	—	2	AAD

AAM	AAM (no operands) ASCII adjust for multiply			Flags O D I T S Z A P C U X X U X U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	83	—	1	AAM

AAS	AAS (no operands) ASCII adjust for subtraction			Flags O D I T S Z A P C U U U X U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	AAS

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

ADC	ADC destination,source Add with carry			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	ADC AX, SI
register, memory	9 + EA	1	2-4	ADC DX, BETA [SI]
memory, register	16 + EA	2	2-4	ADC ALPHA [BX] [SI], DI
register, immediate	4	—	3-4	ADC BX, 256
memory, immediate	17 + EA	2	3-6	ADC GAMMA, 30H
accumulator, immediate	4	—	2-3	ADC AL, 5

ADD	ADD destination,source Addition			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	ADD CX, DX
register, memory	9 + EA	1	2-4	ADD DI, [BX].ALPHA
memory, register	16 + EA	2	2-4	ADD TEMP, CL
register, immediate	4	—	3-4	ADD CL, 2
memory, immediate	17 + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	—	2-3	ADD AX, 200

AND	AND destination,source Logical and			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	AND AL,BL
register, memory	9 + EA	1	2-4	AND CX,FLAG_WORD
memory, register	16 + EA	2	2-4	AND ASCII [DI],AL
register, immediate	4	—	3-4	AND CX,0F0H
memory, immediate	17 + EA	2	3-6	AND BETA, 01H
accumulator, immediate	4	—	2-3	AND AX, 01010000B

CALL	CALL target Call a procedure			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Examples**
near-proc	19	1	3	CALL NEAR_PROC
far-proc	28	2	5	CALL FAR_PROC
memptr 16	21 + EA	2	2-4	CALL PROC_TABLE [SI]
regptr 16	16	1	2	CALL AX
memptr 32	37 + EA	4	2-4	CALL [BX].TASK [SI]

CBW	CBW (no operands) Convert byte to word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	CBW

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

CLC	CLC (no operands) Clear carry flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	CLC

CLD	CLD (no operands) Clear direction flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	CLD

CLI	CLI (no operands) Clear interrupt flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	CLI

CMC	CMC (no operands) Complement carry flag			Flags O D I T S Z A P C X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	CMC

CMP	CMP destination,source Compare destination to source			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	CMP BX, CX
register, memory	9+EA	1	2-4	CMP DH, ALPHA
memory, register	9+EA	1	2-4	CMP [BP+2], SI
register, immediate	4	—	3-4	CMP BL, 02H
memory, immediate	10+EA	1	3-6	CMP [BX].RADAR [DI], 3420H
accumulator, immediate	4	—	2-3	CMP AL, 00010000B

CMPS	CMPS dest-string,source-string Compare string			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	22	2	1	CMPS BUFF1, BUFF2
(repeat) dest-string, source-string	9+22/rep	2/rep	1	REPE CMPS ID, KEY

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

Instruction Set Reference Data (Cont'd.)

CWD	CWD (no operands) Convert word to doubleword			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	5	—	1	CWD

DAA	DAA (no operands) Decimal adjust for addition			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	DAA

DAS	DAS (no operands) Decimal adjust for subtraction			Flags O D I T S Z A P C U X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	DAS

DEC	DEC destination Decrement by 1			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16	2	—	1	DEC AX
reg8	3	—	2	DEC AL
memory	15+EA	2	2-4	DEC ARRAY [SI]

DIV	DIV source Division, unsigned			Flags O D I T S Z A P C U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	80-90	—	2	DIV CL
reg16	144-162	—	2	DIV BX
mem8	(86-96) +EA	1	2-4	DIV ALPHA
mem16	(150-168) +EA	1	2-4	DIV TABLE [SI]

ESC	ESC external-opcode,source Escape			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immediate, memory	8+EA	1	2-4	ESC 6,ARRAY [SI]
immediate, register	2	—	2	ESC 20,AL

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

HLT	HLT (no operands) Halt			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	HLT

IDIV	IDIV source Integer division			Flags O D I T S Z A P C U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	101-112	—	2	IDIV BL
reg16	165-184	—	2	IDIV CX
mem8	(107-118) + EA	1	2-4	IDIV DIVISOR_BYTE [SI]
mem16	(171-190) + EA	1	2-4	IDIV [BX].DIVISOR_WORD

IMUL	IMUL source Integer multiplication			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	80-98	—	2	IMUL CL
reg16	128-154	—	2	IMUL BX
mem8	(86-104) + EA	1	2-4	IMUL RATE_BYTE
mem16	(134-160) + EA	1	2-4	IMUL RATE_WORD [BP] [DI]

IN	IN accumulator,port Input byte or word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, immed8	10	1	2	IN AL, 0FFEAH
accumulator, DX	8	1	1	IN AX, DX

INC	INC destination Increment by 1			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16	2	—	1	INC CX
reg8	3	—	2	INC BL
memory	15 + EA	2	2-4	INC ALPHA [DI] [BX]

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

Instruction Set Reference Data (Cont'd.)

INT	INT interrupt-type Interrupt			Flags O D I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed8 (type = 3)	52	5	1	INT 3
immed8 (type ≠ 3)	51	5	2	INT 67

INTR†	INTR (external maskable interrupt) Interrupt if INTR and IF=1			Flags O D I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	61	7	N/A	N/A

INTO	INTO (no operands) Interrupt if overflow			Flags O D I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	53 or 4	5	1	INTO

IRET	IRET (no operands) Interrupt Return			Flags O D I T S Z A P C R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	24	3	1	IRET

JA/JNBE	JA/JNBE short-label Jump if above/Jump if not below nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JA ABOVE

JAE/JNB	JAE/JNB short-label Jump if above or equal/Jump if not below			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JAE ABOVE_EQUAL

JB/JNAE	JB/JNAE short-label Jump if below/Jump if not above nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JB BELOW

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†INTR is not an instruction, it is included in table 2-21 only for timing information.

JBE/JNA	JBE/JNA short-label Jump if below or equal/Jump if not above			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNA NOT_ABOVE

JC	JC short-label Jump if carry			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JC CARRY_SET

JCXZ	JCXZ short-label Jump if CX is zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	18 or 6	—	2	JCXZ COUNT_DONE

JE/JZ	JE/JZ short-label Jump if equal/Jump if zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JZ ZERO

JG/JNLE	JG/JNLE short-label Jump if greater/Jump if not less nor equal			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JG GREATER

JGE/JNL	JGE/JNL short-label Jump if greater or equal/Jump if not less			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JGE GREATER_EQUAL

JL/JNGE	JL/JNGE short-label Jump if less/Jump if not greater nor equal			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JL LESS

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

JLE/JNG	JLE/JNG short-label Jump if less or equal/Jump if not greater			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNG NOT_GREATER

JMP	JMP target Jump			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	15	—	2	JMP SHORT
near-label	15	—	3	JMP WITHIN_SEGMENT
far-label	15	—	5	JMP FAR_LABEL
memptr16	18 + EA	1	2-4	JMP [BX].TARGET
regptr16	11	—	2	JMP CX
memptr32	24 + EA	2	2-4	JMP OTHER.SEG [SI]

JNC	JNC short-label Jump if not carry			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNC NOT_CARRY

JNE/JNZ	JNE/JNZ short-label Jump if not equal/Jump if not zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNE NOT_EQUAL

JNO	JNO short-label Jump if not overflow			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNO NO_OVERFLOW

JNP/JPO	JNP/JPO short-label Jump if not parity/Jump if parity odd			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JPO ODD_PARITY

JNS	JNS short-label Jump if not sign			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNS POSITIVE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

JO	JO short-label Jump if overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JO SIGNED OVRFLW

JP/JPE	JP/JPE short-label Jump if parity/Jump if parity even			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPE EVEN PARITY

JS	JS short-label Jump if sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JS NEGATIVE

LAHF	LAHF (no operands) Load AH from flags			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	LAHF

LDS	LDS destination,source Load pointer using DS			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers**	**Bytes**	**Coding Example**
reg16, mem32	16 + EA	2	2-4	LDS SI,DATA.SEG [DI]

LEA	LEA destination,source Load effective address			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem16	2 + EA	—	2-4	LEA BX, [BP][DI]

LES	LES destination,source Load pointer using ES			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem32	16 + EA	2	2-4	LES DI, [BX].TEXT BUFF

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

Instruction Set Reference Data (Cont'd.)

LOCK	LOCK (no operands) Lock bus			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	LOCK XCHG FLAG,AL

LODS	LODS source-string Load string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-string (repeat) source-string	12 9 + 13/rep	1 1/rep	1 1	LODS CUSTOMER_NAME REP LODS NAME

LOOP	LOOP short-label Loop			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	17/5	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ short-label Loop if equal/Loop if zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ short-label Loop if not equal/Loop if not zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	19 or 5	—	2	LOOPNE AGAIN

NMI†	NMI (external nonmaskable interrupt) Interrupt if NMI = 1			Flags O S I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†NMI is not an instruction; it is included in table 2-21 only for timing information.

Instruction Set Reference Data (Cont'd.)

MOV	MOV destination,source Move			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
memory, accumulator	10	1	3	MOV ARRAY [SI], AL
accumulator, memory	10	1	3	MOV AX, TEMP_RESULT
register, register	2	—	2	MOV AX,CX
register, memory	8 + EA	1	2-4	MOV BP, STACK TOP
memory, register	9 + EA	1	2-4	MOV COUNT [DI], CX
register, immediate	4	—	2-3	MOV CL, 2
memory, immediate	10 + EA	1	3-6	MOV MASK [BX] [SI], 2CH
seg-reg, reg16	2	—	2	MOV ES, CX
seg-reg, mem16	8 + EA	1	2-4	MOV DS, SEGMENT BASE
reg16, seg-reg	2	—	2	MOV BP, SS
memory, seg-reg	9 + EA	1	2-4	MOV [BX] SEG_SAVE, CS

MOVS	MOVS dest-string,source-string Move string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	18	2	1	MOVS LINE EDIT DATA
(repeat) dest-string, source-string	9 + 17/rep	2/rep	1	REP MOVS SCREEN, BUFFER

MOVSB/MOVSW	MOVSB/MOVSW (no operands) Move string (byte/word)			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	18	2	1	MOVSB
(repeat) (no operands)	9 + 17/rep	2/rep	1	REP MOVSW

MUL	MUL source Multiplication, unsigned			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	70-77	—	2	MUL BL
reg16	118-133	—	2	MUL CX
mem8	(76-83) + EA	1	2-4	MUL MONTH [SI]
mem16	(124-139) + EA	1	2-4	MUL BAUD_RATE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

Instruction Set Reference Data (Cont'd.)

NEG	NEG destination Negate			Flags O D I T S Z A P C X X X X X 1*
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3	—	2	NEG AL
memory	16 + EA	2	2-4	NEG MULTIPLIER

*0 if destination = 0

NOP	NOP (no operands) No Operation			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3	—	1	NOP

NOT	NOT destination Logical not			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3	—	2	NOT AX
memory	16 + EA	2	2-4	NOT CHARACTER

OR	OR destination,source Logical inclusive or			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	OR AL, BL
register, memory	9 + EA	1	2-4	OR DX, PORT ID [DI]
memory, register	16 + EA	2	2-4	OR FLAG BYTE, CL
accumulator, immediate	4	—	2-3	OR AL, 01101100B
register, immediate	4	—	3-4	OR CX,01H
memory, immediate	17 + EA	2	3-6	OR [BX] CMD_WORD,0CFH

OUT	OUT port,accumulator Output byte or word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed8, accumulator	10	1	2	OUT 44, AX
DX, accumulator	8	1	1	OUT DX, AL

POP	POP destination Pop word off stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	8	1	1	POP DX
seg-reg (CS illegal)	8	1	1	POP DS
memory	17 + EA	2	2-4	POP PARAMETER

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

POPF	POPF (no operands) Pop flags off stack			Flags O D I T S Z A P C R R R R R R R R
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	8	1	1	POPF

PUSH	PUSH source Push word onto stack			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
register seg-reg (CS legal) memory	11 10 16 + EA	1 1 2	1 1 2-4	PUSH SI PUSH ES PUSH RETURN CODE [SI]

PUSHF	PUSHF (no operands) Push flags onto stack			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	10	1	1	PUSHF

RCL	RCL destination,count Rotate left through carry			Flags O D I T S Z A P C X X
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1 register, CL memory, 1 memory, CL	2 8 + 4/bit 15 + EA 20 + EA + 4/bit	— — 2 2	2 2 2-4 2-4	RCL CX, 1 RCL AL, CL RCL ALPHA, 1 RCL [BP] PARM, CL

RCR	RCR designation,count Rotate right through carry			Flags O D I T S Z A P C X X
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1 register, CL memory, 1 memory, CL	2 8 + 4/bit 15 + EA 20 + EA + 4/bit	— — 2 2	2 2 2-4 2-4	RCR BX, 1 RCR BL, CL RCR [BX].STATUS, 1 RCR ARRAY [DI], CL

REP	REP (no operands) Repeat string operation			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REP MOVS DEST, SRCE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

REPE/REPZ	REPE/REPZ (no operands) Repeat string operation while equal/while zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPE CMPS DATA, KEY

REPNE/REPNZ	REPNE/REPNZ (no operands) Repeat string operation while not equal/not zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPNE SCAS INPUT_LINE

RET	RET optional-pop-value Return from procedure			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(intra-segment, no pop) (intra-segment, pop) (inter-segment, no pop) (inter-segment, pop)	8 12 18 17	1 1 2 2	1 3 1 3	RET RET 4 RET RET 2

ROL	ROL destination,count Rotate left			Flags O D I T S Z A P C X X
Operands	Clocks	Transfers	Bytes	Coding Examples
register, 1 register, CL memory, 1 memory, CL	2 8 + 4/bit 15 + EA 20 + EA + 4/bit	— — 2 2	2 2 2-4 2-4	ROL BX, 1 ROL DI, CL ROL FLAG_BYTE [DI],1 ROL ALPHA , CL

ROR	ROR destination,count Rotate right			Flags O D I T S Z A P C X X
Operand	Clocks	Transfers*	Bytes	Coding Example
register, 1 register, CL memory, 1 memory, CL	2 8 + 4/bit 15 + EA 20 + EA + 4/bit	— — 2 2	2 2 2-4 2-4	ROR AL, 1 ROR BX, CL ROR PORT_STATUS, 1 ROR CMD_WORD, CL

SAHF	SAHF (no operands) Store AH into flags			Flags O D I T S Z A P C R R R R R
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	SAHF

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer

Instruction Set Reference Data (Cont'd.)

SAL/SHL	SAL/SHL destination,count Shift arithmetic left/Shift logical left			Flags O D I T S Z A P C X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Examples**
register,1	2	—	2	SAL AL,1
register, CL	8+4/bit	—	2	SHL DI, CL
memory,1	15+EA	2	2-4	SHL [BX].OVERDRAW, 1
memory, CL	20+EA+4/bit	2	2-4	SAL STORE COUNT, CL

SAR	SAR destination,source Shift arithmetic right			Flags O D I T S Z A P C X X X U X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2	—	2	SAR DX, 1
register, CL	8+4/bit	—	2	SAR DI, CL
memory, 1	15+EA	2	2-4	SAR N BLOCKS, 1
memory, CL	20+EA+4/bit	2	2-4	SAR N BLOCKS, CL

SBB	SBB destination,source Subtract with borrow			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	SBB BX, CX
register, memory	9+EA	1	2-4	SBB DI, [BX].PAYMENT
memory, register	16+EA	2	2-4	SBB BALANCE, AX
accumulator, immediate	4	—	2-3	SBB AX, 2
register, immediate	4	—	3-4	SBB CL, 1
memory, immediate	17+EA	2	3-6	SBB COUNT [SI], 10

SCAS	SCAS dest-string Scan string			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string	15	1	1	SCAS INPUT_LINE
(repeat) dest-string	9+15/rep	1/rep	1	REPNE SCAS BUFFER

SEGMENT†	SEGMENT override prefix Override to specified segment			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	MOV SS:PARAMETER, AX

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†ASM-86 incorporates the segment override prefix into the operand specification and not as a separate instruction. SEGMENT is included in table 2-21 only for timing information.

Instruction Set Reference Data (Cont'd.)

SHR	SHR destination,count Shift logical right			Flags O D I T S Z A P C X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2	—	2	SHR SI, 1
register, CL	8+4/bit	—	2	SHR SI, CL
memory, 1	15+EA	2	2-4	SHR ID_BYTE [SI] [BX], 1
memory, CL	20+EA+4/bit	2	2-4	SHR INPUT_WORD, CL

SINGLE STEP†	SINGLE STEP (Trap flag interrupt) Interrupt if TF = 1			Flags O D I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

STC	STC (no operands) Set carry flag			Flags O D I T S Z A P C 1
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	STC

STD	STD (no operands) Set direction flag			Flags O D I T S Z A P C 1
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	STD

STI	STI (no operands) Set interrupt enable flag			Flags O D I T S Z A P C 1
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	STI

STOS	STOS dest-string Store byte or word string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string	11	1	1	STOS PRINT_LINE
(repeat) dest-string	9+10/rep	1/rep	1	REP STOS DISPLAY

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†SINGLE STEP is not an instruction. It is included in table 2-21 only for timing information.

Instruction Set Reference Data (Cont'd.)

SUB	SUB destination,source Subtraction			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	SUB CX, BX
register, memory	9 + EA	1	2-4	SUB DX, MATH_TOTAL [SI]
memory, register	16 + EA	2	2-4	SUB [BP + 2], CL
accumulator, immediate	4	—	2-3	SUB AL, 10
register, immediate	4	—	3-4	SUB SI, 5280
memory, immediate	17 + EA	2	3-6	SUB [BP] BALANCE, 1000

TEST	TEST destination,source Test or non-destructive logical and			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	TEST SI, DI
register, memory	9 + EA	1	2-4	TEST SI, END_COUNT
accumulator, immediate	4	—	2-3	TEST AL, 00100000B
register, immediate	5	—	3-4	TEST BX, 0CC4H
memory, immediate	11 + EA	—	3-6	TEST RETURN_CODE, 01H

WAIT	WAIT (no operands) Wait while $\overline{TEST}$ pin not asserted			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3 + 5n	—	1	WAIT

XCHG	XCHG destination,source Exchange			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, reg16	3	—	1	XCHG AX, BX
memory, register	17 + EA	2	2-4	XCHG SEMAPHORE, AX
register, register	4	—	2	XCHG AL, BL

XLAT	XLAT source-table Translate			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-table	11	1	1	XLAT ASCII_TAB

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

Instruction Set Reference Data (Cont'd.)

XOR	XOR destination,source Logical exclusive or			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	XOR CX, BX
register, memory	9 + EA	1	2-4	XOR CL, MASK_BYTE
memory, register	16 + EA	2	2-4	XOR ALPHA [SI], DX
accumulator, immediate	4	—	2-3	XOR AL, 01000010B
register, immediate	4	—	3-4	XOR SI, 00C2H
memory, immediate	17 + EA	2	3-6	XOR RETURN_CODE, 0D2H

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

A Summary of Intel 8086 Memory Addressing Options

MEMORY REFERENCE	SEGMENT REGISTER	BASE REGISTER	INDEX REGISTER	POSSIBLE DISPLACEMENTS: 16-BIT UNSIGNED	8-BIT HIGH ORDER BIT EXTENDED	NONE
NORMAL DATA MEMORY REFERENCE	DS (Alternate* CS, SS or ES)	None	SI	x	x	x
			DI	x	x	x
		BX	SI	x	x	x
			DI	x	x	x
			None	x	x	x
	DS	None	None	x		
	SS (Alternate* CS, DS or ES)	BP	SI	x	x	x
			DI	x	x	x
			None	x	x	
STACK	SS	SP	None			
STRING DATA	ES	None	SI			
			DI			
INSTRUCTION FETCH	CS	PC	None			
BRANCH	CS	PC	None		x	
I/O DATA	DS	DX	None			

* The segment override allows DS or SS to be replaced by one of the other segment registers.

X These are displacements that can be used to compute memory addresses.

Annex C

Table 12.1: The Function Key and Hot Key Commands

Key	Menu Command	Function
F1		Brings up context-sensitive help
F2	Breakpoints/Toggle	Sets breakpoint at cursor position
F3	Window/Close	Closes current window
F4	Run/Go to Cursor	Runs to cursor position
F5		Zooms/unzooms current window
F6	Window/Next	Goes to next window
F7	Run/Trace Into	Executes single source line or instruction
F8	Run/Step Over	Executes single source line or instruction, skipping calls
F9	Run/Run	Runs program
F10		Invokes the main menu bar, takes you out of menus
Alt-F1		Brings up last help screen
Alt-F2	Breakpoints/At	Sets breakpoint at an address
Alt-F3	View/Module	Module pick list
Alt-F4	Run/Animate	Steps continuously updating display
Alt-F5	View/User Screen	Shows your program's screen
Alt-F6	Window/Undo Close	Reopens the last-closed window
Alt-F7	Run/Instruction Trace	Executes a single instruction
Alt-F8	Run/Until Return	Runs until return from function
Alt-F9	Run/Run To	Runs to a specified address
Alt-F10		Invokes the window local menu
Alt-0	Window/Window Pick	Displays a list of all open windows
Alt-1-9		Switch to numbered window
Alt-F		Takes you to the File menu
Alt-V		Takes you to the View menu
Alt-R		Takes you to the Run menu
Alt-B		Takes you to the Breakpoints menu
Alt-D		Takes you to the Data menu
Alt-W		Takes you to the Window menu
Alt-O		Takes you to the Options menu
Alt-X		Quits Turbo Debugger and returns you to DOS
Alt-=	Options/Macros/Create	Defines a keystroke macro
Alt--	Options/Macros/Stop Recording	Ends a macro recording
Ctrl-F2	Run/Program Reset	Stops debug session and resets the program to start again
Ctrl-F4	Data/Evaluate	Evaluates an expression
Ctrl-F7	Data/Watch	Adds a variable to the Watches window
Ctrl-F8	Breakpoints/Toggle	Toggles a breakpoint at cursor
Ctrl-F9	Run/Run	Runs a program
Ctrl-F10		Invokes the window's local menu
Ctrl-Right arrow		Shifts the starting address in a Code, Data, or Stack pane in a CPU window 1 byte up
Ctrl-Left arrow		Shifts the starting address in a Code, Data, or Stack pane in a CPU window 1 byte down
Esc		Closes an Inspector window, takes you out of menus
Ins		Starts text block selection (highlight); use *Left arrow* and *Right arrow* to highlight
Scroll-Lock	Window/Move/Resize	Moves and resizes windows

Table 12.1: The Function Key and Hot Key Commands (continued)

Tab	Moves cursor to next window pane
Shift-Tab	Moves cursor to previous window pane

Commands from the Main Menu Bar

You invoke the main menu bar by pressing the *F10* key; you can also go directly to one of the individual menus by cursoring to the menu title and pressing *Enter* or by pressing the first letter of the menu title. You can also open a menu directly (without first moving to the menu bar) by pressing *Alt* in combination with the first letter of the menu name you desire.

The File Menu

Load	Loads a new program to debug
Change Dir	Changes to new disk and/or directory
Get Info	Displays program info
OS Shell	Starts a DOS command processor
Quit	Returns to DOS

The View Menu

Breakpoints	View breakpoints
Stack	View function-calling stack
Log	View log of events and data
Watches	View variables being watched
Variables	View global and local variables
Module	View program source module
File	View disk file as ASCII or hex
CPU	View CPU instructions, data, stack
Dump	View raw data dump
Registers	View CPU registers and flags
Numeric Processor	View coprocessor or emulator
User Screen	View your program screen
Another	
Module	Makes another Module window
Dump	Makes another Dump window
File	Makes another File window

The Run Menu

Run	Runs your program without stopping
Program Reset	Reloads current program
Go To Cursor	Runs to current cursor location
Trace Into	Executes one source line or instruction
Step Over	Traces, skipping calls
Execute To	Runs to specified address
Until Return	Runs until function returns
Animate	Continuously steps your program
Instruction Trace	Executes a single instruction

The Breakpoints Menu

Toggle	Toggles breakpoint at cursor
At	Sets breakpoint at specified address
Changed Memory Global	Sets global breakpoint on memory area
Expression True Global	Sets global breakpoint on expression
Delete All	Removes all breakpoints

The Data Menu

Inspect	Inspects a data object
Evaluate/Modify	Evaluates an expression
Watch	Adds variable to Watches window
Function Return	Inspects current routine's return value

The Window Menu

Window Pick	Pick window from list of open windows
Next Pane	Goes to next pane in window
Move/Resize	Moves or changes current window size
Close	Erases current window
Undo Close	Undoes last erase command
Dump Pane to Log	Writes current pane to Log window
Restore Standard	Standard window layout
Screen Repaint	Redisplays entire screen

The Options Menu

Language	
Source Module	Sets expression language from source module
C	Uses C for expressions
Pascal	Uses Pascal for expressions
Assembler	Uses assembler for expressions
Macros	
Create	Defines a keystroke macro
Stop Recording	Ends the recording session
Remove	Removes a keystroke macro
Delete All	Removes all keystroke macros
Environment	
Integer Format	Hex/Decimal/Both: Number display format
Display Swapping	None/Smart/Always: User screen swapping mode
Screen Size	25 line/43/50 line: Debugger screen size
Tab Size	Tab width when displaying text files
Path for Source	Directory list for source files
Arguments	Sets program command-line arguments
Save Options	Saves options, macros, windows to disk
Restore Options	Restores options from disk

The Local Menu Commands

You can invoke the pop-up, or "local," menu for the current window by pressing *Alt-F10*. If Control-key shortcuts are enabled, you can go directly to one of the individual menu items by pressing the *Ctrl* key in combination with the first letter of the item you desire. (You can use the installation program TDINST to enable Control-key shortcuts.)

Each type of window (Breakpoint, Module, etc.) and each pane within a window has a different local menu. The following sections describe the local menu for each window and pane.

Some panes have shortcuts to commonly used commands on their local menu. In the following section, these special keys are listed before the menu commands for the pane to which they apply. In many panes, the *Enter* key is a shortcut to examining or changing the currently highlighted item. The *Del* key often invokes the local menu command that deletes the highlighted item. Some panes let you start typing letters or numbers

without first invoking a local menu command. In these cases, the prompt box for one of the local menu items pops up to accept your input.

The Breakpoints Window Local Menu

The Breakpoints window has two panes, the List pane on the left, and the Detail pane on the right. Only the List pane has a local menu.

Set Action	
Break	Sets breakpoint to stop program
Log	Sets breakpoint to log an expression
Execute	Sets breakpoint to execute an expression
Condition	
Always	Unconditional breakpoint
Changed Memory	When memory area changes
Expression True	When an expression is true
Hardware	
Cycle Type	
Read Memory	Match memory reads
Write Memory	Match memory writes
Access Memory	Match memory read or write
Input I/O	Match I/O input
Output I/O	Match I/O Output
Both I/O	Match I/O input or output
Fetch Instruction	Match instruction fetch
Address	
Above	Match above an address
Below	Match below an address
Range	Match within address range
Not Range	Match outside address range
Less or Equal	Match below or equal to address
Greater or Equal	Match above or equal to address
Equal	Match a single address
Unequal	Match all but a single address
Match All	Match any address
Data	
Above	Match above a value
Below	Match below a value
Range	Match within a range of values
Not Range	Match outside a range of values
Less or Equal	Match below or equal to value
Greater or Equal	Match above or equal to value

Equal	Match a single value
Unequal	Match all but a single value
Match All	Match all values
Pass Count	Number of times to skip breakpoint
Enable/Disable	Toggles breakpoint enabled
Add	Adds a new breakpoint
Global	Adds a new global breakpoint
Remove	Removes highlighted breakpoint
Delete All	Deletes all breakpoints
Inspect	Looks at code where this breakpoint is set

The CPU Window Menus

The CPU window has five panes, each with a local menu: the Code pane, the Data pane, the Stack pane, the Register pane, and the Flags pane.

The Code Pane Local Menu

Goto	Displays code at new address
Origin	Displays code at *cs:ip*
Follow	Displays code at JMP or CALL target
Caller	Displays code at calling function
Previous	Displays code at last address
Search	searches for instruction or bytes
View Source	Switches to Module window
Mixed	No/Yes/Both: Mixes source code with disassembly
New CS:IP	Sets *CS:IP* to execute at new address
Assemble	Assembles instruction at cursor
I/O	
In Byte	Reads a byte from an I/O location
Out Byte	Writes a byte to an I/O location
Read Word	Reads a word from an I/O location
Write Word	Writes a word to an I/O location

Typing any character is a shortcut for the Remove local menu command in this pane.

The Data Pane Local Menu

Goto	Displays data at new address
Search	Searches for string or data bytes

Next	Searches again for next occurrence
Change	Changes data bytes at cursor address
Follow	Follows near pointer chain
Long Follow	Follows far pointer chain
Previous	Displays data at last address
Display As	
Byte	Displays hex bytes
Word	Displays hex words
Long	Displays hex 32-bit long words
Comp	Displays 8-byte Pascal comp integers
Float	Displays short (4-byte) floating numbers (Pascal singles)
Real	Displays 6-byte floating-point numbers (Pascal reals)
Double	Displays 8-byte floating-point numbers
Extended	Displays 10-byte floating-point numbers (C long double)
Block	
Clear	Sets memory block to zero
Move	Moves memory block
Set	Sets memory block to value
Read	Reads from file to memory
Write	Writes from memory to file

Typing any character is a shortcut for the Change local menu command in this pane.

The Stack Pane Local Menu

Goto	Displays stack at new address
Origin	Displays data at SS:SP
Follow	Displays code pointed to by current item
Previous	Restores display to last address
Change	Allows you to edit information

Typing any character is a shortcut for the Change local menu command in this pane.

The Register Pane Local Menu

Increment	Adds one to highlighted register
Decrement	Subtracts one from highlighted register
Zero	Clears highlighted register

Change	Sets highlighted register to new value
Registers 32-bit	No/Yes: Toggles 32-bit register display

Typing any character is a shortcut for the Change local menu command in this pane.

The Flags Pane Local Menu

Toggle	Sets or clears highlighted flag

Pressing *Enter* is a shortcut for the local menu command in this pane.

The File Window Menu

The File window shows the contents of the disk file as hex bytes or as a disk file.

Goto	Displays line number or hex offset
Search	Searches for string or data bytes
Next	Searches again for next occurrence
Display As	Ascii/Hex: Set file display mode
File	Switches to view new file
Edit	Edits file or changed bytes at cursor

Typing any character is a shortcut for the Search local menu command.

The Log Window Menu

The Log window shows messages sent to the log.

Open Log File	Starts logging to a file
Close Log File	Stops logging to a file
Logging	No/Yes: Toggles logging
Add Comment	Writes user comment to log
Erase Log	Clears all log messages

Typing any character is a shortcut for the Add Comment local menu command.

The Module Window Menu

The Module window shows the source file for the program module.

Inspect	Shows contents of variable under cursor
Watch	Adds variable under cursor to watch list
Module	Changes to display different module
File	Changes to display different file
Previous	Displays last module and position
Line	Displays line number in module
Search	Searches for text string
Next	Searches for next occurrence of string
Origin	Displays current program location
Goto	Shows source or instructions at address
Edit	Starts editor to edit source file

Typing any character is a shortcut for the Goto local menu command.

The Numeric Processor Window Menus

The Numeric Processor window has three panes: The Register pane, the Status pane and the Control pane.

The Register Pane Local Menu

The following keys are shortcuts to local menu commands in this pane:

Zero	Clears the highlighted register
Empty	Sets the highlighted register to empty
Change	Sets the highlighted register to a value

Typing any character is a shortcut for the Change local menu command in this pane

The Status Pane Local Menu

The following keys are shortcuts to local menu commands in this pane:

Toggle	Cycles through valid flag values

Pressing *Enter* is a shortcut for the local menu command in this pane.

The Control Pane Local Menu

Toggle	Cycles through valid flag values

Pressing *Enter* is a shortcut for the local menu command in this pane.

The Stack Window Menu

The Stack window shows the currently active functions.

Inspect	Shows source code for highlighted function
Locals	Shows argument types for function

Pressing *Enter* is a shortcut for the Inspect local menu command.

The Variables Window Menus

The Variables window has two panes, each with a local menu: The Global Symbol pane and the Local Symbol pane.

The Global Symbol Pane Local Menu

Inspect	Shows contents of highlighted symbol
Change	Changes value of highlighted symbol

Pressing *Enter* is a shortcut for the Inspect local menu command in this pane.

The Local Symbol Pane Local Menu

Inspect	Shows contents of highlighted symbol
Change	Changes value of highlighted symbol

Pressing *Enter* is a shortcut for the Inspect local menu command in this pane.

The Watches Window Menu

The Watches window has a single pane that shows the names and values of the variables you're watching.

Watch	Adds a variable to watch
Edit	Lets you edit a variable
Remove	Deletes highlighted variable
Delete All	Deletes all watch variables
Inspect	Shows contents of highlighted variable
Change	Changes contents of highlighted variable

The following keys are shortcuts to local menu commands in this window:

Key	Function
any character	Watch
Enter	Watch

The Inspector Window Local Menu

An Inspector window shows the contents of a data item.

Range	Selects array members to inspect
Change	Changes the value of highlighted item
Inspect	Opens new Inspector for highlighted item
Descend	Expands highlighted item into this Inspector
New Expression	Inspects a new expression in this Inspector

Text Panes

This is the generic name for a pane that displays the contents of a text file. The blinking cursor shows your current position in the file. The following table lists all the commands.

Key	Function
Ins	Marks text block
Up arrow	Moves up one line
Down arrow	Moves down one line
Right arrow	Moves right one column
Left arrow	Moves left one column
Ctrl-Right arrow	Moves to next word
Ctrl Left arrow	Moves to previous word
Home	Goes to start of line
End	Goes to last character on line
PgUp	Scrolls up one screen
PgDn	Scrolls down one screen
Ctrl-Home	Goes to top line of pane
Ctrl-End	Goes to bottom line of pane
Ctrl-PgUp	Goes to first line of file
Ctrl PgDn	Goes to last line of file

If you are not using the Control-key shortcuts, you can also use the WordStar-style Control keys for moving around a Text pane:

Key	Function
Ctrl-S	Moves left one column
Ctrl-D	Moves right one column
Ctrl-E	Moves up one line
Ctrl-X	Moves down one line
Ctrl-R	Scrolls up one screen
Ctrl-C	Scrolls down one screen
Ctrl-F	Moves to next word
Ctrl-A	Moves to previous word

List Panes

This is the generic name for a pane that lists information you can scroll through. A highlight bar shows your current position in the list. Here's a list of all the commands available to you.

Key	Function
Up arrow	Moves up one item
Down arrow	Moves down one item
Home	Goes to start of line
End	Goes to last character on line
PgUp	Scrolls up one screen
PgDn	Scrolls down one screen
Ctrl-Home	Goes to top line of list pane
Ctrl-End	Goes to bottom line of list pane
Ctrl-PgUp	Goes to first item in list
Ctrl-PgDn	Goes to last item in list
Backspace	Backs up one character in incremental match
Letter	Incremental search (select by typing)

You can also use the WordStar-style Control keys for moving around a List pane:

Key	Function
Ctrl-E	Moves up one line
Ctrl-X	Moves down one line
Ctrl-R	Scrolls up one screen
Ctrl-C	Scrolls down one screen

Commands in Prompt Boxes

The following table shows the commands available when you're inside a prompt box.

Key	Function
Up arrow	Moves up one history item
Down arrow	Moves down one history item
Right arrow	Moves right one character
Left arrow	Moves left one character
Ctrl-Right arrow	Moves to next word
Ctrl-Left arrow	Moves to previous word
Home	Goes to start of line
End	Goes to last character on line
PgUp	Scrolls up one screen
PgDn	Scrolls down one screen
Ctrl-Home	Goes to top line of list pane
Ctrl-End	Goes to bottom line of list pane
Ctrl-PgUp	Goes to first item in list
Ctrl-PgDn	Goes to last item in list
Backspace	Deletes the character before the cursor
Enter	Accepts your input and proceed
Del	Deletes the character after the cursor
Esc	Cancels the prompt and returns to menu

Window Movement Commands

Key	Function
Scroll Lock	Toggles window-positioning mode
Up arrow	Moves window up one line
Down arrow	Moves window down one line
Right arrow	Moves window right one column
Left arrow	Moves window left one column
Shift-Up arrow	Moves window up several lines
Shift-Down arrow	Moves window down several lines
Shift-Right arrow	Moves window right several columns
Shift-Left arrow	Moves window left several columns
Home	Moves to left side of screen
End	Moves to right side of screen
PgUp	Moves to top line of screen
PgDn	Moves to bottom line of screen
Enter	Accepts current position
Esc	Cancels window-positioning command

Wildcard Search Templates

You can use wildcard search templates in two circumstances:

- when entering a file name to load or examine
- when entering a text search expression in a text pane

The ? (question mark) matches any single character in the search expression. The * (asterisk) matches 0 or more characters in the search expression.

File Lists

When you are prompted for a file name and you supply one of the following responses, you will get a file list to pick from:

- a file name containing the * or ? wildcard characters
- a disk drive letter, like C:
- a directory name, like /MYDIR

The list of files has two panes. On the left appears a list of the files in the directory you specified. On the right appears a list of the directories in the

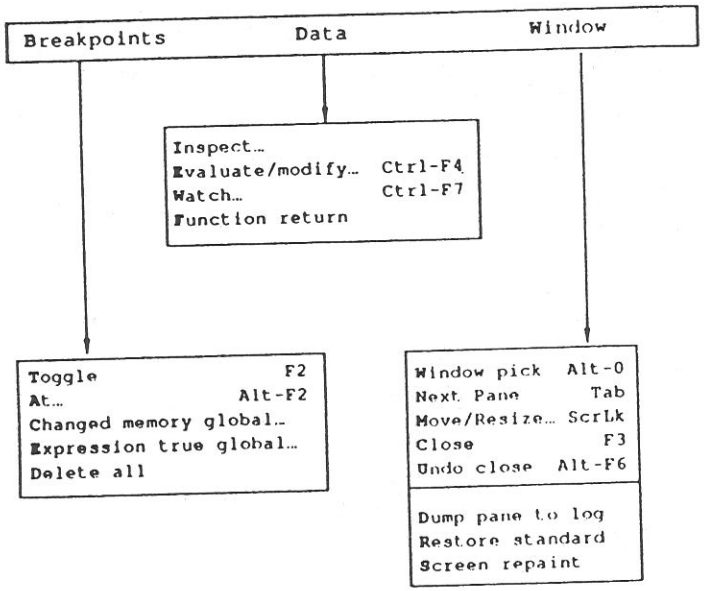


Figure 12.2: The Breakpoints, Data, and Window Menus

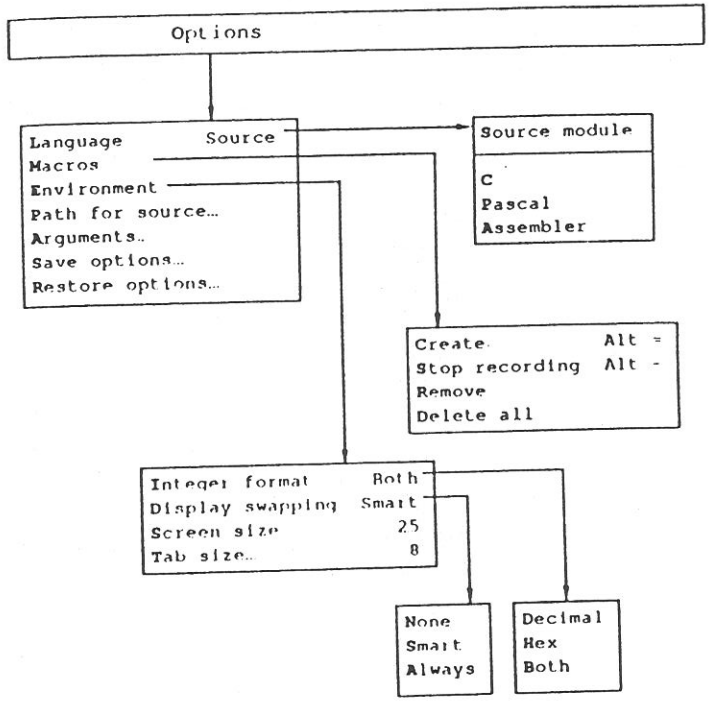


Figure 12.3 The Options Menu

directory you specified, along with the special entry "..\" indicating the parent directory. You can use *Tab* and *Shift-Tab* to switch between the two panes, just as with other multi-paned windows.

By pressing *Enter* while the highlight is on a directory in the right pane, you switch to that directory. The matching files in that directory then appear in the left pane.

If you want to enter a new directory or wildcard file name, you can press *Ins* and edit or replace the current directory and wildcard mask.

Remember that a file list is like any other list, meaning that you can incrementally match on a file name by starting to type the name of the file that you want. When the highlight is over the correct file name, press *Enter* to accept the file name.

Complete Menu Tree

Figures 12.1, 12.2, and 12.3 show the complete structure of Turbo Debugger's pull-down menus.

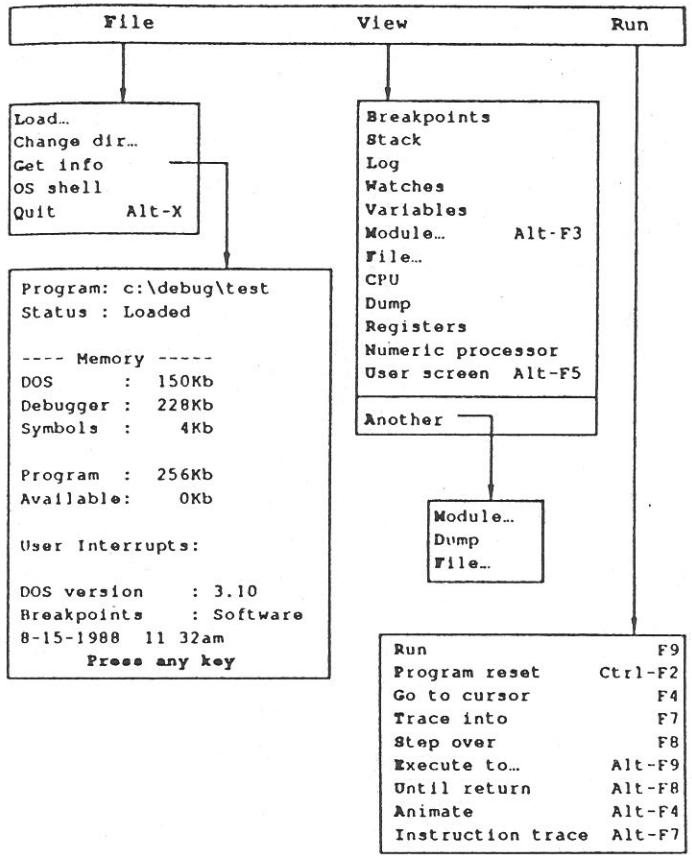


Figure 12.1. The File, View, and Run Menus

Annex 4.

Taula ASCII.

ASCII

Dec	Hex	CHR	Dec	Hex	CHR	Dec	Hex	CHR
000	00	NUL	041	29	)	082	52	R
001	01	SOH	042	2A	*	083	53	S
002	02	STX	043	2B	+	084	54	T
003	03	ETX	044	2C	,	085	55	U
004	04	EOT	045	2D	—	086	56	V
005	05	ENQ	046	2E	.	087	57	W
006	06	ACK	047	2F	/	088	58	X
007	07	BEL	048	30	0	089	59	Y
008	08	BS	049	31	1	090	5A	Z
009	09	HT	050	32	2	091	5B	[
010	0A	LF	051	33	3	092	5C	\
011	0B	VT	052	34	4	093	5D	]
012	0C	FF	053	35	5	094	5E	^
013	0D	CR	054	36	6	095	5F	_
014	0E	SO	055	37	7	096	60	`
015	0F	SI	056	38	8	097	61	a
016	10	DLE	057	39	9	098	62	b
017	11	DC1	058	3A	:	099	63	c
018	12	DC2	059	3B	;	100	64	d
019	13	DC3	060	3C	<	101	65	e
020	14	DC4	061	3D	=	102	66	f
021	15	NAK	062	3E	>	103	67	g
022	16	SYN	063	3F	?	104	68	h
023	17	ETB	064	40	@	105	69	i
024	18	CAN	065	41	A	106	6A	j
025	19	EM	066	42	B	107	6B	k
026	1A	SUB	067	43	C	108	6C	l
027	1B	ESCAPE	068	44	D	109	6D	m
028	1C	FS	069	45	E	110	6E	n
029	1D	GS	070	46	F	111	6F	o
030	1E	RS	071	47	G	112	70	p
031	1F	US	072	48	H	113	71	q
032	20	SPACE	073	49	I	114	72	r
033	21	!	074	4A	J	115	73	s
034	22	"	075	4B	K	116	74	t
035	23	#	076	4C	L	117	75	u
036	24	$	077	4D	M	118	76	v
037	25	%	078	4E	N	119	77	w
038	26	&	079	4F	O	120	78	x
039	27	'	080	50	P	121	79	y
040	28	(	081	51	Q	122	7A	z

Salts condicionals

Nombre	Descripción	Equiv.	Condición
JA	· Bifurcar si superior.	—	CF = 0 y ZF = 0
JAE	· Bifurcar si superior o igual.	—	CF = 0
JB	· Bifurcar si inferior.	—	CF = 1
JBE	· Bifurcar si inferior o igual.	—	CF = 1 o AF = 1
JC	• Bifurcar si acarreo.	JB	CF = 1
JCXZ	· Bifurcar si CX es cero.	—	CX = 0
JE	· Bifurcar si igual.	—	ZF = 1
JG	· Bifurcar si mayor.	—	ZF = 0 y SF = OF
JGE	· Bifurcar si mayor o igual.	—	SF = OF
JL	· Bifurcar si menor.	—	SF <> OF
JLE	· Bifurcar si menor o igual.	—	ZF = 1 o SF <> OF
JNA	• Bifurcar si no superior.	JBE	CF = 1 o AF = 1
JNAE	• Bifurcar si no superior ni igual.	JB	CF = 1
JNB	• Bifurcar si no inferior.	JAE	CF = 0
JNBE	• Bifurcar si no inferior ni igual.	JA	CF = 0 y ZF = 0
JNC	• Bifurcar si no acarreo.	JAE	CF = 0
JNE	· Bifurcar si no igual.	—	ZF = 0
JNG	• Bifurcar si no mayor.	JLE	ZF = 1 o SF <> OF
JNGE	• Bifurcar si no mayor ni igual.	JL	SF <> OF
JNL	• Bifurcar si no menor.	JGE	SF = OF
JNLE	• Bifurcar si no menor ni igual.	JG	ZF = 0 y SF = OF
JNO	· Bifurcar si no desbordamiento	—	OF = 0
JNP	· Bifurcar si no paridad.	—	PF = 0
JNS	· Bifurcar si no signo/si positivo.	—	SF = 0
JNZ	• Bifurcar si no cero.	JNE	ZF = 0
JO	· Bifurcar si desbordamiento.	—	OF = 1
JP	· Bifurcar si paridad.	—	PF = 1
JPE	• Bifurcar si paridad par.	JP	PF = 1
JPO	• Bifurcar si paridad impar.	JNP	PF = 0
JS	· Bifurcar si signo.	—	SF = 1
JZ	• Bifurcar si igual a cero.	JE	ZF = 1

Para bifurcar si	Números sin signo	Números con signo
destino < fuente	JB	JL
destino = fuente	JE	JE
destino > fuente	JA	JG
destino <= fuente	JBE	JLE
destino >= fuente	JAE	JGE
destino <> fuente	JNE	JNE